NASA CONTRACTOR REPORT 166383

Application of Control Theory to Dynamic Systems Simulation

D. M. Auslander
R. C. Spear
G. E. Young

NASA Cooperative Agreement NCC 2-67
August 1982

NASA

CELSS

NASA CONTRACTOR REPORT 166383

Application of Control Theory to Dynamic Systems Simulation

D. M. Auslander
R. C. Spear
G. E. Young
University of California at Berkeley
Berkeley, California

Prepared for Ames Research Center under
NASA Cooperative Agreement NCC 2-67

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

TABLE OF CONTENTS

PREFACE

This report contains 3 papers which consider the application of control theory to dynamic systems simulation. These articles contain theory and methodology applicable to controlled ecological life support systems (CELSS). Discussed are spatial effects on system stability, design of control systems with uncertain parameters, and an interactive computing language (PARASOL-II) designed for dynamic system simulation, report-quality graphics, data acquisition, and simple real-time control.

Spatial Effects on the Stability of a Food-Limited Moth Population

D. M. Auslander, Professor
Mechanical Engineering Department
University of California, Berkeley

Abstract

Having obtained anomolous results in an attempt to continue the simulation study of moth-wasp interaction in Auslander, Oster and Huffaker (1974), attention centered on the role of spatial heterogeneity in an environment with the moth (Anagasta Kuhniella) alone. Because of the probable presence of chaotic components in the population behavior (random appearing behavior that is actually caused by deterministic influences) a statistically-based parameter sensitivity and parameter identification method was used. By defining a binary performance criterion that measured the ability of a model with a specific set of parameters to maintain a stable population, the importance of spatial heterogeneity was confirmed. In addition, the use of Monte-Carlo type simulation studies, combined with a binary performance criterion was demonstrated to be effective for parameter identification and/or parameter sensitivity determination of at least some systems with chaotic or nearly chaotic behavior.

Acknowledgements

The author would like to acknowledge the help of Dr. Harold T. Gordon, Department of Entomology and Parasitology, Univ. of Calif., Berkeley, who supplied information used in the formulation of the food utilization and development portions of the model. This work was supported in part by NASA, Life Sciences Division, Ames Research Center, Agreement No. NCC2-67.

Introduction

In a series of experiments, Huffaker, White and Hassel (White and Huffaker, 1969 a and b, Hassel and Huffaker, 1969) observed the behavior of populations of the moth Anagasta Kühniella under controlled laboratory conditions. The purpose of the original experiments was to study the effects of predation and parasitism in the control of Anagasta population. In this study, however, the only data used was that drawn from experiments in which the moth population was maintained alone, with no predators or parasites present. These were experiments that were used as controls by Huffaker et al.

The experiments consisted of maintaining populations of Anagasta in environmentally controlled spaces (about 0.1 m^3 in one set of experiments and about 14 m^3 in the other set). Food was available in containers located in the test cell. For the most part, larvae lived in these containers. Measured quantities of food were introduced on a regular basis by replacing old food containers with new ones. The primary, and most reliable, data sets obtained from the experiments were biweekly counts of dead adult moths for the duration of the experiments (some were as long as three years). Other data, such as estimates of unused food, estimates of living adults, estimates of larval and pupal mortality, were also obtained in some instances. Figure 1 shows a typical experimental result, the count of dead adults (normalized to a per day figure) as a function of time. This figure contains almost the same information as in Fig. 3 of White and Huffaker, 1969a, Ecosystem IV-2, however, it was replotted from White's original laboratory records and also includes information from the early portion of the experiment that was omitted from the published data.

A common purpose of simulation studies is to test hypotheses concerning the significant and relevance of certain aspects of a system with respect to

the behavior of that system under specified conditions. The hypotheses are expressed in the form of a model, in many cases, a mathematical model. Validation of such a model consists of some form of comparison of simulated results to experimental results. The results of these studies are to increase confidence in the predictive abilities of the model (if the validation was successful), often with the hope that it can ultimately be used as a policy-making aid (as in pest management, for example), and/or the suggestion of new hypotheses and thus new experiments that have the potential of leading to future increases in confidence. An alternative statement of purpose for simulation/modeling work is that it is a means of extracting the maximum possible amount of information from experimental (or field) data. Several previous studies are relevant to this one with respect to the kinds of hypotheses that they were testing. Hassel and Huffaker (1969) used key factor analysis to construct a model capable of predicting generation-to-generation populations of the two-species system consisting of Anagasta and the parasitic wasp, Venturia canescens. Podoler (1974b) used a similar analytic technique to produce a model of a similar experimental system, consisting of the moth Plodia interpunctella and the same parasitic wasp. Auslander, Oster and Huffaker (1974) used a partial differential equation model to test the hypothesis that the generational discretization observed in the experiments could be predicted from a model based primarily on physiological data concerning the species involved.

This study began as an extension to the simulation work (Auslander, Oster, Huffaker) cited above. In particular, it was an attempt to seek explanations for the population crash that appeared just before the synchronized host/parasite generations started. This crash is common to the simulated and laboratory data and the thought was that it might somehow be an essential part

of the synchronization process. That study, however, is still to be done. In the course of starting that work anamolous results were obtained that, ultimately, led to the present paper.

The same partial differential equation structure used by Auslander, Oster and Huffaker (1974) was used in the new study. A major change, though, was made in the way in which mortality was computed. Rather than using a density dependent larval mortality, as was used in the above models, it was postulated that a more realistic larval mortality model would be nutrition-dependent. A model was thus constructed in which larval moth mortality was made a function of body weight, which, in turn, depended on the amount of food available to the larvae as it grew. This constituted a delayed, density dependent mortality, but the amount of the delay was strongly dependent on the density also. Using this model, it was found that no reasonable parameter set would lead to a solution in which the moth, by itself, would reach a stable population. Instead, as shown by the sample response in Fig. 2, the population oscillated wildly, quite unlike the behavior shown in the laboratory and illustrated by Fig. 1. The postulate that spatial heterogeneity in the population, which had been ignored in the previous models, might be important appeared promising and thus lead to this present study of the factors affecting the moth population itself.

At the same time, questions concerning mechanisms for validating population models, or extracting information from experimental data, were being considered in the light of theoretical indications that oscillatory population data could well contain "chaotic" components (Oster, Auslander and Allen, 1976). These components have the same appearance as noise in data records, but arise from deterministic sources. These components call into question the usual validation procedure of "matching" the simulated and experimental data because such matching can never be achieved in the chaotic regime. On the

other hand, if the randomly-appearing components in experimental data could have come from deterministic sources, they can contribute to the process of validation or information extraction. For this reason, it was decided to concentrate model fitting and validation procedures on gross behavioral characteristics of the population and use the Monte Carlo simulation method (repeated simulations with randomly chosen parameter sets) developed by Spear and Hornberger (1980) to establish parameter significance.

Dynamic Population Model

Model Structure

The model structure used for this study follows the pattern set in Auslander, Oster and Huffaker (1974). The relevant result from that paper is recapitulated here, and the specific adaptations made to that model are given in detail. The generalized von Foerster equation (Eq. 2.6) is

$$\frac{\partial n(t,\underline{\xi})}{\partial t} + \Sigma \frac{\partial}{\partial \xi_i} [g_i(t,\underline{\xi})\, n(t,\underline{\xi})] = -\mu(t,\underline{\xi})\, n(t,\underline{\xi}) \qquad (1)$$

where $n(t,\underline{\xi})$ is the population density function,

t is time,

$\underline{\xi}$ is a vector of physiological properties (age, body weight length,..)

$g_i = \frac{d\xi}{dt}$ is the growth rate for the i-th property, and

μ is the mortality function.

For the Anagasta model, the physiological properties of age and dry body weight will be used. With the definitions,

$\xi_1 = z$ = age

$g_1(t,\underline{\xi}) = g_z(t,z,w) = 1$ for uniform chronological aging

$\xi_2 = w$ = dry body weight

$g_2(t,\underline{\xi}) = g_w(t,z,w)$ = weight gain (or loss) function,

then, $n(t,\underline{\xi}) = n(t,z,w)$, and Eq. (1) becomes

$$\frac{\partial n(t,z,w)}{\partial t} + \frac{\partial n(t,z,w)}{\partial z} + \frac{\partial}{\partial w}\left[g_w(t,z,w)\, n(t,z,w)\right] = -\mu(t,z,w)\, n(t,z,w) \quad (2)$$

The birth process is the boundary condition for Eq. (2) and is represented by an integral over the egg-laying ages,

$$n(t,z=0,w) = \int_{\alpha}^{\alpha+\gamma} m(t,z',w)\, n(t,z',w)\, dz' \quad (3)$$

where m is the fecundity function.

Equation (2) is a partial differential equation with three independent variables, t, z, and w. Since there are no convolution-type terms in Eq. (2), for computational purposes it is necessary to retain a two-dimensional representation of n(t,z,w) in memory, representing the instantaneous population density as a function of age and weight. A major computational simplification can be obtained if the restriction that all newly-laid eggs have the same weight is imposed. Because Eq. (2) has no dispersive (second-order) terms, if the projection of the population density onto the w-axis for z=0 is an impulse function (all newly-laid eggs have the same weight), that impulse function will not disperse. Thus the projection of the population density at any age, z, will also be an impulse. A reduction in dimensionality can be obtained by eliminating the w-dimension and, instead, introducing the variable W(t,z) that represents the weight of an individual at time t and of age z. The population density, n(t,z,w) then becomes n(t,z), and Eq. (2) can be written as the following set of coupled equations,

$$\frac{\partial n(t,z)}{\partial t} + \frac{\partial n(t,z)}{\partial z} = -\mu(t,z,W)\ n(t,z) \tag{4}$$

$$\frac{\partial W(t,z)}{\partial t} + \frac{\partial W(t,z)}{\partial z} = -g_w(t,z,W)\ w(t,z) \tag{5}$$

For computational purposes, instead of retaining a two-dimensional function in memory, Eqs. (4) and (5) require the retention of two one-dimensional functions, the instantaneous population density as a function of age, and the instantaneous weight function as a function of age. It should be noted that although the restriction that all newly-laid eggs have the same weight has been applied, that restriction can be relaxed somewhat as long as all newly-laid eggs at any time t have the same weight.

Mortality

The mortality has been assumed to be related to body weight, particularly for the larvae. Defining a survivorship,

$$s = 1 - \mu = s(z,W) \tag{6}$$

where μ is the mortality function from Eq. (2), a relationship can be postulated between the survivorship and normalized body weight, WNORM. In the absence of any further definitive information on the nature of this function, a two-parameter family of curves has been used, Fig. 3. The normalized body weight is the actual body weight, W, divided by the weight that would have been attained at that age if unlimited amounts of food were available, $W_{max}(z)$. The curves are crudely sigmoidal in shape and are described by the parameters WNZ, the value at which the curve leaves the WNORM axis, and H, the slope. The curve is limited at the top to a value of 1.

Body Weight

The rate-of-change of body weight is assumed to be proportional to the excess (or deficiency) of food consumed over the metabolic food needs. Because only one kind of food was used in the experiments (Zoom, a rolled wheat), it is not necessary to break the food down into more basic nutritional components. The basic body weight model is thus,

$$\frac{dW}{dt} = b_1(f - f_m) \tag{7}$$

where W is dry body weight, mg

f is the food consumption rate per individual, mg/day,

f_m is the metabolic food requirement for an individual, mg/day,

and b_1 is a constant.

The metabolic food requirement is assumed to be proportional to the body weight,

$$f_m = b_2 W \tag{8}$$

where b_2 is a constant. Reasonable ranges for the parameters b_1 and b_2 were selected to be consistent with data in Waldbauer (1968).

Food Supply and Consumption

The total amount of food available at any time is described by the conservation equation,

$$\frac{dF}{dt} = f_s - f_c^T \tag{9}$$

where F is the amount of food available, mg,

f_s is the food supply rate, mg/day,

f_c^T is the total food consumption rate, mg/day, summed over all individuals,

$$f_c^T = \Sigma f \tag{10}$$

where f is the food consumption rate for an individual, mg/day.

The actual food consumption rate is governed by the nature of competition among individuals, since in these experiments the larval food supply was the limiting population growth factor. The assumption made here is the very simple one that each individual gets the same percentage of its maximum consumption rate, where the maximum consumption rate is defined as the amount that it would eat if unlimited amounts of food were available. This is a competition model of uniform success as a percentage of demand, with the maximum consumption rate regarded as the demand. The demand for an individual is broken into two components, a portion to support growth, f_{dg}, and a portion to meet the metabolic needs, f_{dm},

$$f_d = f_{dg} + f_{dm} \tag{11}$$

where f_d is the food demand for an individual, mg/day.
As noted in Eq. (8), the metabolic demand is assumed proportional to the body weight. The growth-related demand is assumed to be a function of age only,

$$f_{dg} = f(z) \tag{12}$$

The assumptions of uniform success in competition for food, and growth-related demand equal to a function of age alone are extremely simplistic, but are all that can be justified at present. (In fact, the function of age is set equal to a constant in the simulations).

The actual consumption is related to the demand through a saturating function. A parameter having units of time, T_f, is defined by,

$$T_f = F/f_d^T \tag{13}$$

where f_d^T is the total food demand rate, mg/day, that is, the food demand rate summed over all individuals,

$$f_d^T = \Sigma f_d \tag{14}$$

The total actual food consumption, f^T, is computed by,

$$f^T = (1 - e^{-aT_f})\, f_d^T \tag{15}$$

where "a" is a constant. The actual food consumption for an individual, f, is computed from the equal-percentage-allocation assumption,

$$f = (f^T/f_d^T)\, f_d \tag{16}$$

That is, the ratio of consumption to demand for each individual is the same as the ratio of total consumption to total demand for the whole population.

Fecundity

Podoler (1974a) has shown that the number of eggs per female in the Indian Meal-moth (Plodia Interpunctella) is a function of the adult body weight. Following this observation, the number of eggs per female, m, is taken as a product of functions based on body weight, age, and a maximum feasible egg-laying rate,

$$m = f_1(\hat{W})\, f_2(z)\, m_{max}$$

where $\hat{W} = W/W_{max}$ is the normalized body weight. The body-weight function is taken as a power-law form,

$$f_1(W) = W^p \tag{18}$$

The curve shapes for p less than one, equal to one and greater than one are shown in Fig. 4. Values of p greater than one are indicated by Podoler's data (1974a, Fig. 5). The age-dependent function, $f_2(z)$ was taken equal to one for all females of egg-laying age.

Development Time

The amount of time required for development from egg to adult can vary considerably (Auslander, Oster, Huffaker, 1974), with crowding given as a major factor in increasing the required development time. Introduction of varying development time in a rigorous manner would dictate the addition of second-order terms to the basic population equation, Eq. (2), to account for dispersion in the population. To avoid the large increase in computing effort and memory that would be necessary for simulation of the second-order equation an approximation to true dispersive behavior was employed. In this approximation, the transition between the larval and pupal states is assumed to be a function of body weight, larger larvae "graduating" earlier. The body weights of the pupal cohorts thus formed are taken to be the average of all those larvae that went to form the cohort. A two-parameter family of curves is used to determine whether or not a given individual will become a pupa. A typical curve is shown in Fig. 5. The curve consisting of the sloping line plus the segments of the $\hat{W} = 1$ and $\hat{W} = 0$ as shown form a transition boundary. The transition zone of N days is the period at the end of the larval stage during which transition is possible. The transition boundary indicates the weight required for graduation and is a decreasing function of age. That is, as larvae get older they become eligible for the transition to pupae at a lower body weight. The two parameters describing the curve are WSL, the slope of the curve (a positive WSL implies a negative slope) and WZ the ordinate corresponding to N/2.

Physical Environment

The food containers used to supply new food were not explicitly considered in any of the previous simulation studies of this population. These food containers were introduced periodically, full of fresh food, to replace the containers that had been in the controlled space the longest. Replacements were done once a week; one set of experiments worked on a 25 week cycle (that is, containers were replaced after being in the system for 25 weeks) and the other worked on a 17 week cycle. The model used of this environment was that the eggs, larvae and pupae were completely restricted to the container that they were born into. Larval mobility was neglected. Adults were assumed to have equal access to all containers for egg-laying (and equal preference). The complete model, including the spatial heterogeneity due to the discrete food containers, consisted of one replication of the model discussed above for each container or set of containers. The model components were completely non-interactive except for the coupling due to egg-laying, which was assumed equally distributed over all containers.

Monte Carlo Simulations

The Monte Carlo simulation method described by Spear and Hornberger (1980) can be used to determine the sensitivity of a certain system behavior to given parameters and, for those parameters for which a significant sensitivity is found, it can be used to determine the expected range of the parameters. It can also be used to find multivariate sensitivities, that is, system behavioral patterns that are affected by two or more parameters changing together. The separation of parameter sets is based on the application of a binary performance criterion to the simulated system behavior. Each free parameter in the system is characterized by a distribution giving its likely or possible range of

values. By free parameter is meant any parameter that is to be tested. In some cases, sensitivity information might be desired for some parameter that is reasonably well known from a priori information, while, in other cases, a probable range of values is desired for a more-or-less unknown parameter. It thus serves as a combination of parameter identification and sensitivity determination method. For each simulation, a value is picked for each parameter from its specified distribution and the simulated behavior is computed.

The selection function is the key to use of this Monte Carlo method. It is applied to the set of simulation results to separate it into two sets: a set of results that exhibit a pre-defined behavior (that set is called B) and a set of results that does not exhibit that behavior (called NOT-B, $\bar{B}$). By analyzing the parameter sets associated with B and $\bar{B}$, information can be obtained about correlations between areas in the parameter space and the probability of producing B or $\bar{B}$. Where correlations are strong, inferences can be made about the role of the particular parameter or combination of parameters in determining the behavior of the model being studied. Weak correlations imply either that the behavior of the model is not sensitive to the parameter (or combination) or that the coverage of the parameter space was not dense enough. Acceptance of the null hypothesis (no influence) in examining the B and $\bar{B}$ sets thus leads to a subjective judgement as to whether the original Monte Carlo experiment was adequate.

Ideally, a study of this sort would be conducted by defining the model, including the distributions for the parameters; defining B; specifying the number of trials and statistical procedures; performing the experiments; and analyzing the data. In practice, this is usually impossible because not enough is known about the model's behavior to keep the number of trials

required (and thus the cost) of the study to a reasonable level. Since the design of subsequent experiments draws on the results of previous ones, independence, and thus confidence in the significance of the results, is increasingly lost. As with all modeling/simulation projects, care must be taken to avoid producing self-fulfilling results.

The statistical tests used are primarily non-parametric tests to determine whether or not distributions separate. Spear and Hornberger (1980) used the Kolmorov-Smirnoff test; because of the discrete nature of some of the parameters used, the chi-squared test is used here. The basic separation test determines the univariate properties of the system, that is, the role played by individual parameters in determining the nature of the model's response. Parameters can act in combination, however, and such multivariate dependence is harder to determine. For example, in a process containing two chemicals that can react, the concentrations of the two chemicals will only affect the product when they are present together. Univariate statistics would show little influence for each of the species, but bivariate statistics would show a strong influence when both are present. In this study, linear bivariate dependence is tested by computing all possible correlation coefficients. Linear high-order dependence can also be tested, although it has not been done in this study. Nonlinear dependencies are much harder to find.

A major hypothesis in the development of this parameter sensitivity/identification method is that many systems of interest can be characterized by behavior criteria that are relatively simple. In fact as the system gets more complex, it may be easier to characterize it with a binary performance criterion, particularly in highly-evolved, natural systems. It is the ability to use a binary performance criterion that makes the method economically attractive.

Selection of Test Behavior and Free Parameters

The original motivation for this study, the inability to achieve stability in the first modeling efforts, provided the major behavior criterion: stability. The quantitative definition of stability used was the ratio of the root-mean-square of the deviation of the adult population from the mean to the mean population. The evaluation was carried out over the simulated time period corresponding to the time period for which stable populations were observed in the experiment. Further limitation was placed in defining B by putting bounds on acceptable values for the mean - this was most important in eliminating cases for which the population crashed giving a mean of zero.

The free parameters were selected from a combination of hypothesis-testing and parameter identification concerns. The prime hypothesis arising from the initial round of simulation experiments was that spatial heterogeneity was a major determinant of dynamic behavior in the original experiment. The parameter characterizing the system's spatial properties is the number of food containers. By making the number of food containers a free parameter, the sensitivity of the system behavior to the number of containers can be tested. As a verification of the reasonableness of the model, the range of values indicated as leading to stable behavior should include the value actually used in the experiments.

The other free parameters were selected from the model parameters described above. Most of these parameters were unknown, so the Monte Carlo procedures were used to perform parameter identification. Wherever possible, reasonable ranges for parameters were obtained from the biological literature. For computational simplicity, all parameters were described by uniform distributions.

Results

The general procedure used in performing the Monte Carlo test runs was to start with wide parameter variations to get a feeling for areas in the parameter space that show promise of yielding interesting solutions. In many instances, after examining the response curves for regions suggested by the initial Monte Carlo runs, portions of the model were restructured. The distributions used for the parameters and the number of trials were successively readjusted until results with statistical significance were obtained. Throughout this process, a persistent characteristic of the responses of the model was observed for virtually all parameter combinations that resulted in solutions with any modicum of stability. This was an oscillatory component of the solution with a period of 6 to 8 days. Figure 6 is a typical response showing this component. It was initially felt that this solution component might be an artifact of the solution method or model structure. Changes were made first in discretization intervals for the numerical solution, then the sharp corners in the mortality functions were modified, all to no avail. Finally, a spectral analysis was made of the original White data using a discrete Fourier transform (FFT), giving the spectrum shown in Fig. 7. The peak at 7 days is distinct. Caution must be observed in interpreting that information, however, because the original data was obtained by sampling the population only twice a week, usually at alternating 3 and 4 day intervals. In order to get a data set suitable for the FFT program, White's data was interpolated to give a data set with even spacing of samples.

The partial differential equations describing the population evolution were solved by discretizing the age dimension into cohorts and discretizing time (see Auslander, Oster, Huffaker, 1974). The time discretization interval and the age discretization interval were equal. Spatial heterogeneity was included by solving "n" such sets of partial differential equations simultaneously,

to account for "n" feeding tins. The equations were coupled only through the birth process. The computation time increases with the square of the inverse of the discretization interval because the number of time steps per unit time increases with the inverse of the discretization interval and so does the number of cohorts. The amount of memory required is proportional to the inverse of the discretization interval. Both computation time and memory requirements increase in proportion to the number of feeding tins.

Most of the simulations were done using a discretization interval of two days. This was a compromise used to increase the efficiency of the Monte Carlo process because of the square-law dependence of computing time on discretization interval. Smaller intervals made some changes in specific simulation responses, but made no significant changes in the test behavior function.

The final density distributions and the ranges of values used for the free parameters are shown in Fig. 8. The two questions associated with these results are: 1) are the separations statistically significant, and 2) do the simulated results obtained by using parameters drawn from the "passed" region of the parameter space match the actual experimental results (i.e., has the fitting procedure been successful in the conventional modeling sense). The statistical results are shown in Tables I and II. A most important result is the separation shown for the number of feeding tins, verifying the original hypothesis that stability was dependent on spatial heterogeneity. Variables with low values for the chi-squared statistic probably have relatively little influence on the nature of the solution within the range of values studied. The qualification is added to that statement because the possibility always exists that there is a very strong, very narrow influence that was completely missed because not enough trials were made. Table II gives all possible correlation coefficients.

None of the correlation coefficients is high enough to warrant further investigation of bivariate or higher-order dependencies at this point.

The second question, the match of the simulated and experimental responses, is addressed in Figures 6 and 9. These figures are based on models with parameters selected from inside the "passed" areas. The simulated results reproduce several important salient features of the experimental results shown in Fig. 1. In particular, the responses begin with a violent transient and then settle down to stable behavior with a high frequency oscillatory component. The experimental results also exhibit low frequency oscillations that are not usually present in the simulated results.

Of the parameters examined here, half were found to have a strong influence on stability, and the other half had little or no influence. In particular, the parameters H and WNZ used to define the mortality relation had the strongest influence. Referring back to Fig. 8, the passed regions favored high values for H (the slope) and low values for WNZ (the starting point of the rise in survivorship, see Fig. 3). Physiologically, this indicates that, based on the model presented here, survival remains relatively high until the body weight gets to be much lower than normal, at which point survival drops off rapidly. The next highest influence came from the parameter p, used to determine the shape of the birthrate/body weight curve (Fig. 4). As predicted by Podoler, values of p greater than one were favored. Finally, as noted above, stability was associated with large numbers of food tins.

The parameters relating to body weight, b_1 and b_2, and the parameters describing the laval/pupal transition (WZ and WSL) had relatively little influence on population stability.

Some further insights on the role of various parameters in determining key aspects of the system response can be made by calculating correlation

coefficients between the various system parameters and the mean of the population for "passed" cases. These results are also shown in Table II.

Conclusions

First, the results indicate that the spatial arrangement within the experimental environment played a very strong role in the determination of the dynamic behavior of the moth population. The conclusion, based on model studies, is significant because it arises from complex interactions among system variables and cannot be deduced from qualitative or intuitive analysis of the system. It is not even apparent that a correct qualitative judgement could have been made as to whether the introduction of spatial heterogeneity in this system tends to stabilize or destabilize it. Further study is required to determine if the strong role played by the spatial arrangement carries over to the predator/host and parasite/host situations studied by White and Huffaker, for which the moth-alone systems were the controls. In any case, the conclusion reached here should be considered in the design of future experiments of this type and in the interpretation of past experiments. For example, the experiment described by Podoler (1974a,b) provides an environment with much greater larval mobility than the Huffaker/White experiments. Although no control (moths only) experiments were reported for that study, the results reached here indicate a strong likelihood that the responses would not have reached a stable behavior if the larvae were able to take advantage ofthe barrier-free environment.

The second conclusion concerns the utility of the Monte Carlo method with a binary decision function as a technique for parameter identification and sensitivity analysis. The role of the number of food tins has been inferred with a reasonable degree of confidence even though, in the conventional modeling

sense, no simulated response was ever produced that could be quantitatively compared to the experimental results on a point-by-point basis. Given the complexity of any system involving living organisms, and given the uncertainty as to the role of stochastic influences (because of the possibilities of chaotic behavior), it seems likely that no such point-to-point comparison could ever be achieved. The use of a statistically-based identification/ sensitivity method is essential in such problems, and this particular case, the method used worked effectively. A further observation, however, is that the type of information obtained from this technique gives a much stronger feeling for applicable parameter ranges than either identification procedures or sensitivity analyses based on point-by-point comparisons and could be an effective adjunct to such techniques in problems to which they can be applied.

References

1. Auslander, D. M., G. F. Oster, C. B. Huffaker (1974), "Dynamics of Interacting Populations," Journal of the Franklin Institute, 297, 345-375.

2. Hassel, M. P. and C. B. Huffaker (1969), "Regulatory Processes and Population Cyclicity in Laboratory Populations of Anagasta Kühniella (Zeller) (Lepidoptera: Phycitidae) II. Parasitism, Predation, Competition and Protective Cover," Res. Popul. Ecol XI, 150-185.

3. Oster, G. F., D. M. Auslander, T. T. Allen (1976), "Deterministic and Stochastic Effects in Population Dynamics," Trans. ASME, Journal of Dynamic Systems, Measurement and Control, 98, 44-48.

4. Podoler, H. (1974a), "Effects of Intraspecific Competition in the Indian Meal-Moth (Plodia Interpunctella Hubner) (Lepidoptera Phycitidae) on Populations of the Moth and its Parasite Nemeritis Canescens (Gravenhorst) (Hymenoptera: Ichneumonidae)," J. Anim. Ecol.

5. Podoler, H. (1974b), "Analysis of Life Tables for a Host and Parasite (Plodia-Nemeritis) Ecosystem," J. Anim. Ecol.

6. Spear, R. C. and G M Hornberger (1980), "Eutrophication in Peel Inlet-II. Identification of Critical Uncertainties via Generalized Sensitivity Analysis," Water Research, Vol 14, pp. 43-49.

7. Waldbauer, G. P. (1968),"The Consumption and Utilization of Food by Insects," pp. 229-288, v 5, Advances in Insect Physiology (edited by J. W. L. Beament, J. E. Treherne and V. B. Wigglesworth).

8. White, E. G. and C. B. Huffaker (1969a), "Regulatory Processes and Population Cyclicity in Laboratory Populations of Anagasta Kühniella (Zeller) (Lepidoptera: Phycitidae) I. Competition for Food and Predation," Res. Popul. Ecol. XI, 57-83.

9. White, E. G. and C. B. Huffaker (1969b), "Regulatory Processes and Population Cyclicity in Laboratory Populations of Anagasta Kuhniella (Zeller) (Lepidoptera· Phycitidae) II. Parasitism, Predation, Competition and Protective Cover," Res. Popul. Ecol XI, 150-185.

Table I - Chi-Squared Statistics (degrees-of-freedom = 9)

Parameter	Pass/Fail		Pass/All		Fail/All	
	chi-sq	confidence	chi-sq	confidence	chi-sq	confidence
b1	8.2	0.5	3.5	0.03	2.1	0.01
b2	10.2	0.6	4.3	0.07	2.6	0.01
H	90	0.995+	39	0.995+	24	0.995+
WNZ	76	0.995+	33	0.995+	21	0.98
p	52	0.995+	24	0.99	13	0.8
NGH	45	0.995+	19	0.96	12	0.7
WZ	14.7	0.9	6.1	0.2	3.9	0.05
WSL	7.2	0.3	3.0	0.02	1.9	0

Table II - Correlation Coefficients

		All Trials	Passed	Failed
1	2	4.380E-02	8.934E-02	1.265E-02
1	3	-8.893E-02	-1.100E-01	-7.690E-02
1	4	8.520E-02	6.351E-02	9.897E-02
1	5	1.730E-02	1.504E-01	-5.408E-02
1	6	-6.497E-03	-3.376E-02	1.895E-02
1	7	2.729E-02	9.464E-02	-2.146E-02
1	8	-1.585E-02	-8.586E-02	3.480E-02
1	9	4.168E-01	5.987E-01	3.706E-01
2	3	1.274E-03	-2.276E-02	7.719E-02
2	4	-2.503E-02	3.943E-02	-1.289E-01
2	5	-6.780E-02	-6.916E-02	-3.052E-02
2	6	1.776E-02	-3.691E-02	9.611E-02
2	7	5.745E-02	-1.835E-02	8.864E-02
2	8	-9.295E-02	-7.613E-02	-1.054E-01
2	9	-2.079E-01	-2.282E-01	-1.619E-01
3	4	-1.191E-01	9.316E-02	9.200E-03
3	5	9.720E-03	5.954E-02	-2.320E-01
3	6	7.334E-04	-4.867E-02	-1.745E-01
3	7	-2.376E-02	3.163E-02	4.790E-02
3	8	7.403E-02	8.227E-02	7.565E-02
3	9	4.750E-01	2.933E-01	4.379E-01
4	5	1.132E-02	1.231E-01	1.510E-01
4	6	7.753E-02	2.379E-01	1.884E-01
4	7	6.894E-02	1.128E-01	-5.841E-02
4	8	6.004E-02	8.764E-04	1.168E-01
4	9	-4.749E-01	-3.844E-01	-3.669E-01
5	6	4.328E-02	-2.117E-01	6.797E-02
5	7	-9.451E-03	8.948E-02	3.194E-03
5	8	-6.610E-03	4.745E-02	-4.839E-02
5	9	1.184E-01	1.451E-01	-1.380E-01
6	7	1.447E-02	4.023E-02	7.358E-02
6	8	-3.271E-02	1.754E-02	-7.599E-02
6	9	-6.897E-02	-2.146E-01	-2.235E-01
7	8	-5.860E-02	2.095E-02	-1.198E-01
7	9	-1.444E-01	-4.159E-02	-1.324E-01
8	9	3.598E-02	-3.729E-02	1.119E-01

Codes for Parameters:

1	b1
2	b2
3	H
4	WNZ
5	p
6	NGH
7	WZ
8	WSL
9	Population mean

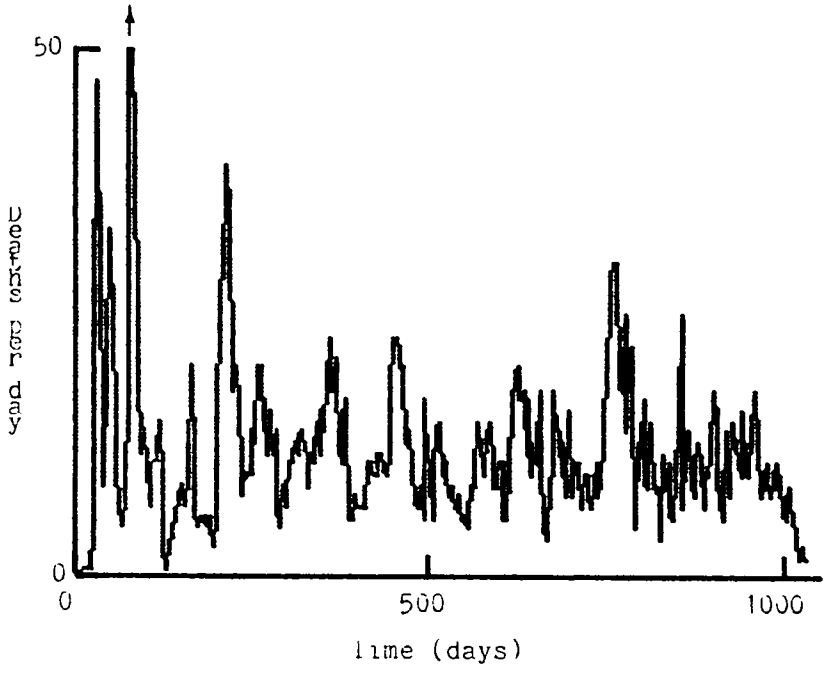


Figure 1. White and Huffaker (1969a) Data for Ecosystem IV-2

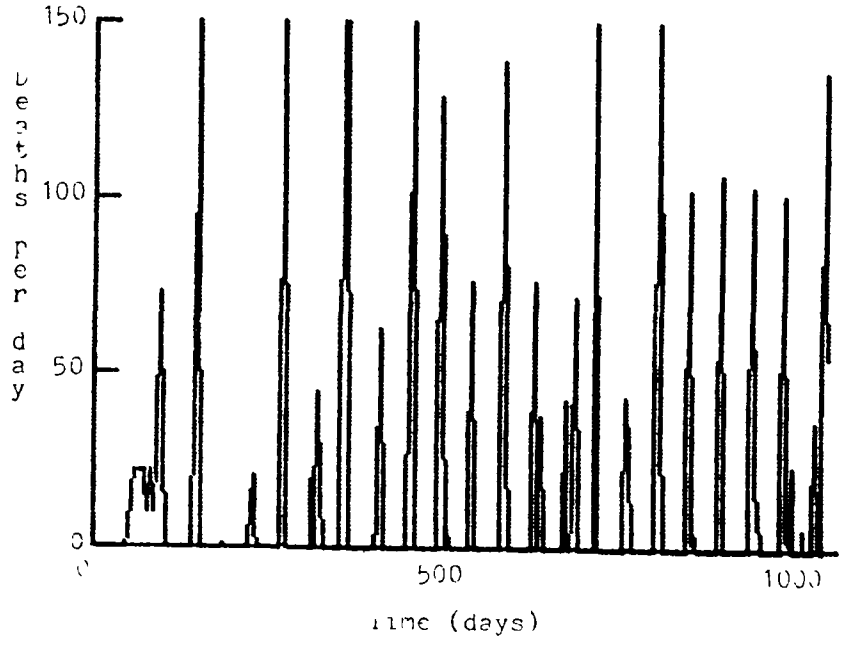


Figure 2. Response of Preliminary Model

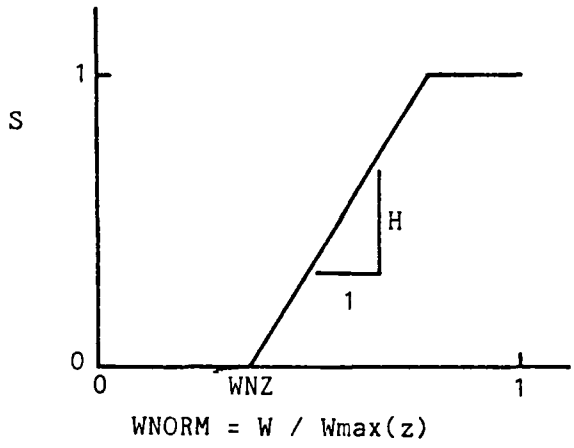


Figure 3. Survivorship Function

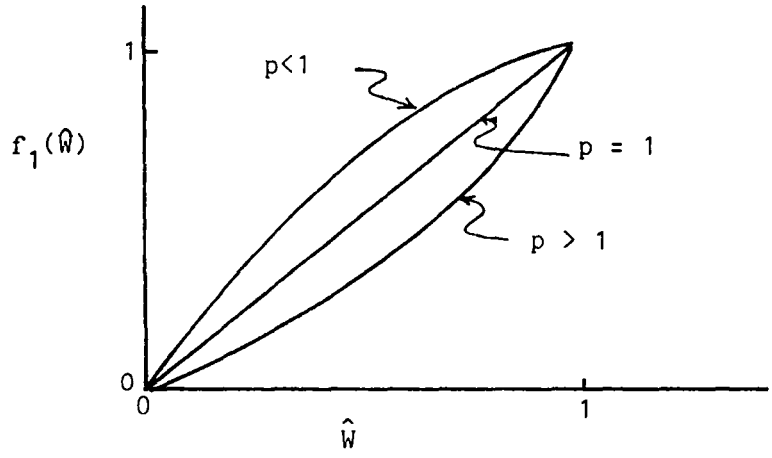


Figure 4. Fecundity-Body Weight Function

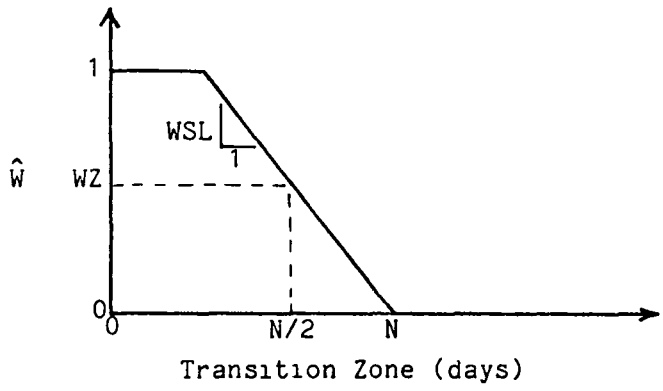


Figure 5. Criterion for Larval/Pupal Transition

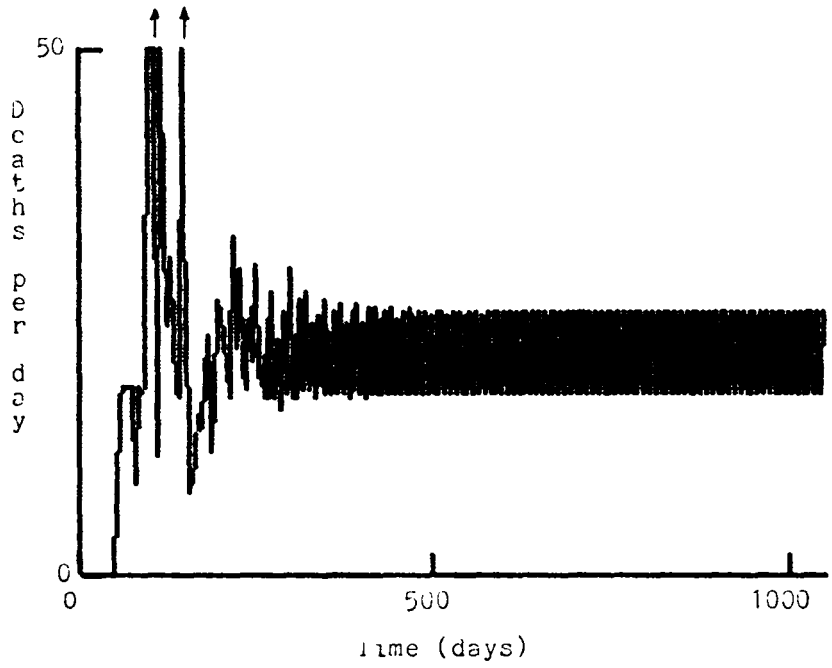


Figure 6. Simulated Model Response

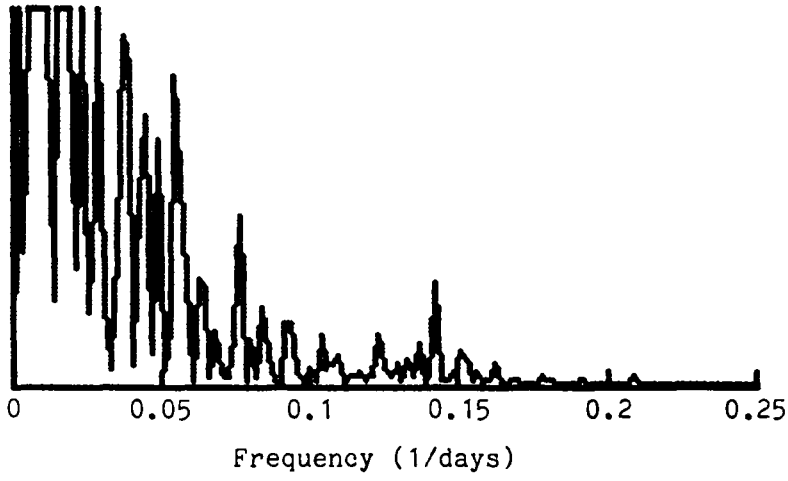


Figure 7. Spectral Analysis of White's Data

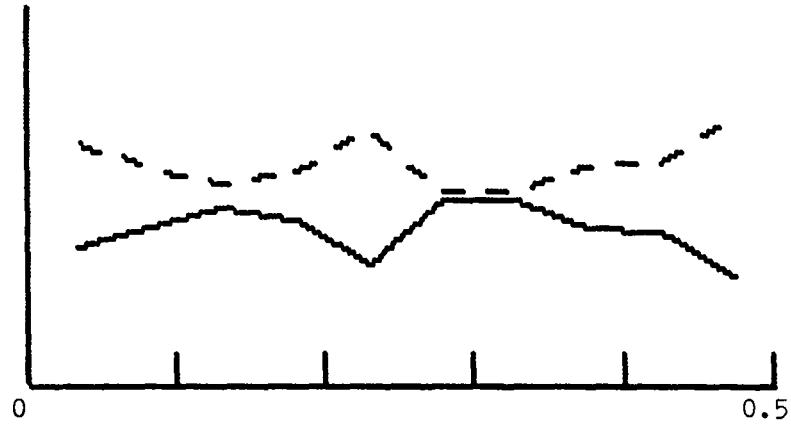
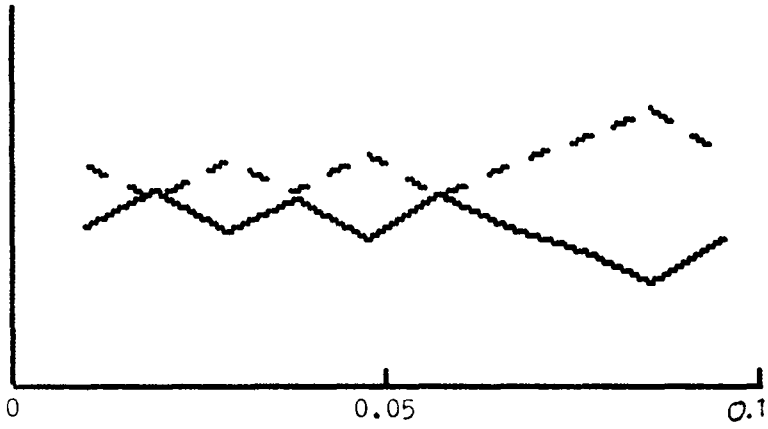


Figure 8. Density Distributions
(passed - solid; failed - dashed)



b2 (parameter in body weight)

Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

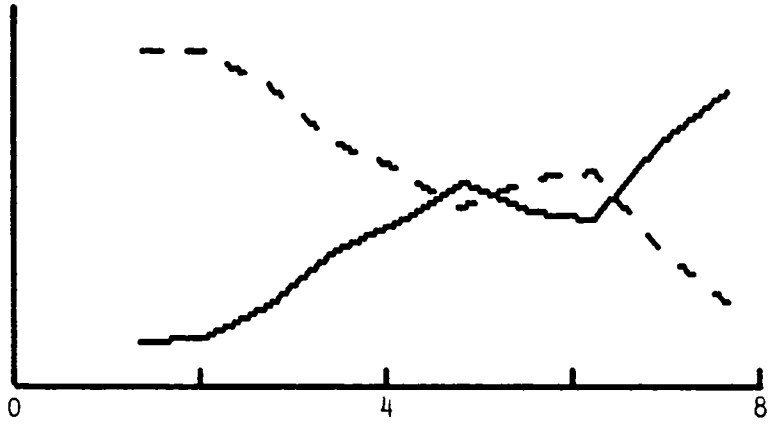


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

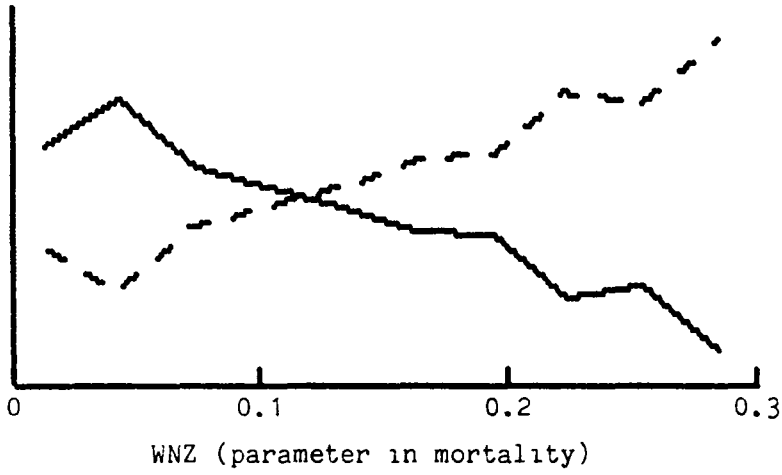


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

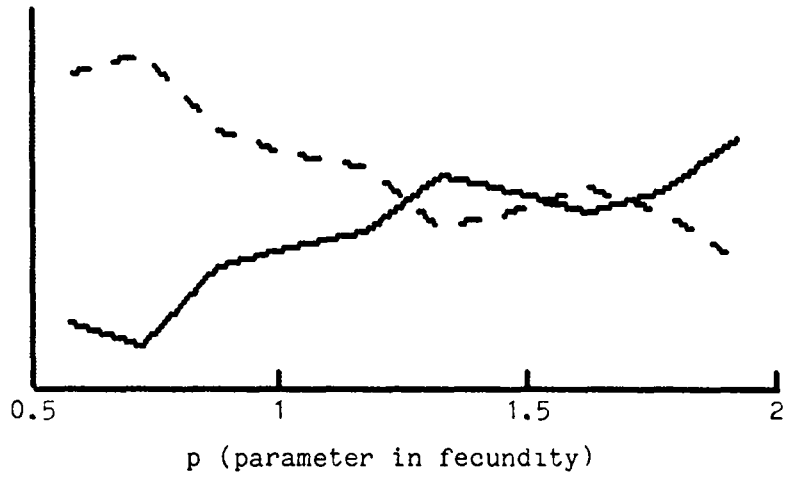


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

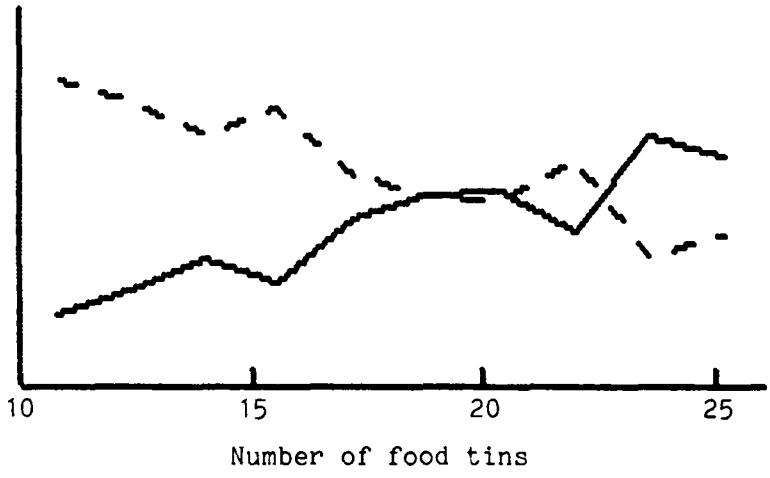


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

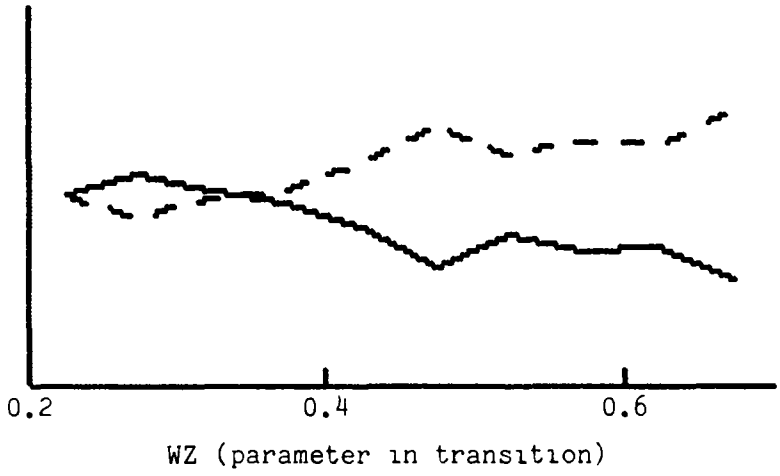


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

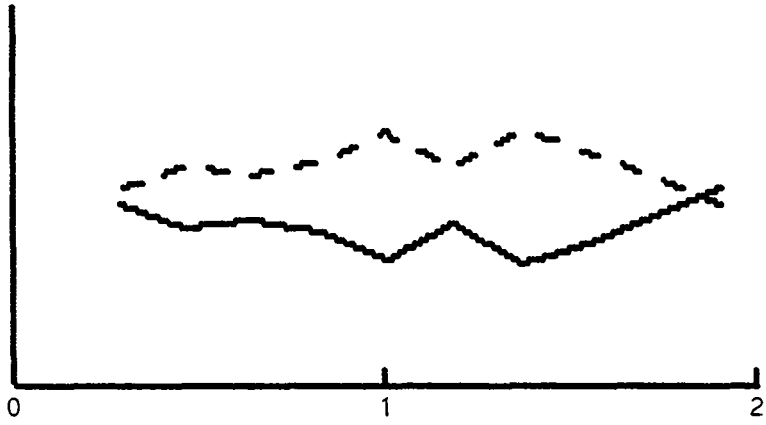


Figure 8. Density Distributions (con't)
(passed - solid; failed - dashed)

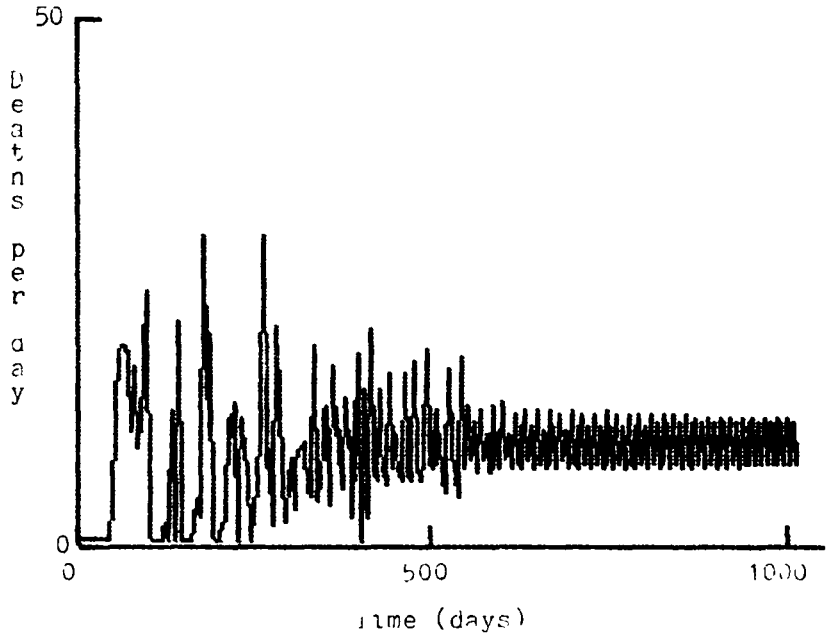


Figure 9. Simulated Model Response, parameters in passed zone.

Design of Control Systems with Uncertain Parameters

by

David M. Auslander
Robert C. Spear
Gary E. Young

ABSTRACT

A design method for control systems with uncertain parameters is presented. The method utilizes a generalized sensitivity approach which separates the parameter space into regions which produce a system response that satisfies given design criteria and regions which do not. Nonparametric statistics and confidence limits for the binomial distribution are used to determine degree of parameter sensitivity and to locate regions in the parameter space which maximize the probability of producing a desirable system response. In an example it is shown that a given parameter may have to be known to a lesser degree of uncertainty to be able to specify a satisfactory design.

INTRODUCTION

With increased interest in the design of nonlinear control systems and control systems with uncertain parameters it is desirable to have a design method or strategy which does not depend upon the linearity of the system. Conventional control design strategies, however, are applicable only to linear systems or to systems which have been linearized in one way or another. For the case of uncertain parameters, the system response will no longer be deterministic and must be interpreted in some sort of probabilistic manner.

Here, we advocate the idea of combining statistics and simulation for the analysis and/or design of linear or nonlinear dynamic systems with uncertain parameters.

Tiwari and Hobbie [1] used this concept to analyze an aquatic ecosystem. They assigned Gaussian probability distributions to all uncertain parameters of their system model and then used a simulation procedure to obtain the sample means and standard deviations of the variables of interest as functions of time. As was pointed out in their paper, the results were quite different than those obtained by using the mean values of the uncertain parameter distributions and performing only one 'deterministic' simulation.

Spear [2] used nonparametric statistics to investigate the stability of a monopropellant rocket engine model in which the dynamic parameters were uncertain. He randomly selected values from a-priori parameter distributions and directly calculated the minimum negative real axis crossing of the frequency locus in the Nyquist plane. The cumulative distribution of the minimum negative real axis crossing was obtained and Kolmogorov-Rényi statistics applied to this resulting distribution. Thus, given the uncertainties of the dynamic parameters, a measure of confidence that the rocket engine model would be stable was acquired.

Spear and Hornberger [3] studied a model of an ecosystem with uncertain parameters using a generalized sensitivity analysis. Fundamental to their analysis was the concept of a problem-defining behavior. This behavior, denoted B, is a pattern of state variable response that mimicked the qualitative behavior of the real ecosystem they were studying in a given manner. Uniform probability distributions were assigned to all uncertain parameters. A randomly selected set of parameter values, called a parameter vector, $\underline{\xi}$, was then used to simulate the system. The resulting response either did exhibit the behavior or did not exhibit the behavior, B. A new set of parameters was then selected and the process repeated. Since only the parameter vector, $\underline{\xi}$, differed from one simulation run to the next it was possible to conduct a series of runs and to accumulate two sets of parameter vectors, those which gave rise to the behavior and those which did not. Nonparametric statistics were then used to determine if the distributions of the parameters in these parameter vectors separate under the behavioral classification. Thus, for the given model, to what extent the behavior, B, was sensitive to each of the uncertain parameters could be determined.

In this paper we investigate the use of simulation and statistics as a tool for control system design. In particular, we explore the utility of this simulation-based approach in the design of conventional control systems. As expected, it will be seen that this approach has its greatest utility in designing systems for which conventional methods fail, i.e., for nonlinear systems with uncertain parameters. However, it will be shown that this approach also has advantages over conventional techniques in designing linear control systems with fixed parameters.

The method will be examined in the following section. Two examples are given for which conventional design techniques are compared with the simulation-based method. In these examples the plant parameters are assumed known and fixed.

In the last section we consider the design control systems for which conventional techniques are not applicable. The method is illustrated through an example of the design of a controller for a continuous-stirred tank reactor. Two of the reactor parameters associated with the reaction rate are treated as unknown with known bounds. We also discuss the results obtained from this example when the parameters are considered fixed in any region within the original bounds.

METHOD FOR FIXED PLANT PARAMETERS

Classical design techniques, such as the Root Locus Method, can be used to design control systems which are linear with fixed plant parameters. Design criteria are generally given in terms of a damping ratio (damping per cycle) and eigenvalue placement. However, these are indirect design criteria. What is of prime interest is the time response of the controlled system. Unless we are able to solve for the motion, knowledge of pole placement and damping ratio does not give us knowledge of this time response.

If we are to specify the design criteria by some measure of the time response then we must be able to verify their satisfaction (or non-satisfaction) for given values of the control parameters. Here we propose using simulation as an experimental method for determining satisfaction of the design criteria.

For a control system with fixed plant parameters, the only uncertain parameters are the combination of controller gains which will simultaneously meet the design criteria. Thus, the task of the designer is to locate regions in the parameter space where these conditions will be satisfied.

Using this simulation-based approach, the procedure is as follows:

1. Assign probability distributions to all of the unknown control gain parameters. If there is no information to indicate otherwise, take these distributions to be uniform. The designer will usually have at least an order of magnitude estimate as to what the proper range of the control gains should be. If this is not the case a few simulations using 'extreme' values for the gains will determine a range such that combinations of extreme values produce undesirable system response.
2. Randomly select a set of controller parameters from the a-priori distributions assigned in Step 1 and simulate the system. Record values of the parameters and the subsequent measure (or measures) of the system response at the end of each 'run.'
3. Repeat Step 2 to obtain a significant number of trials. For this case, 'significant' is subjective. In his search for regions in the parameter space yielding satisfactory system response the designer must always compromise between desired density coverage of the space (which is infinite) and allowable computation time.
4. After the simulation runs are complete, for each criterion, separate the controller parameters into two groups, those which produced a system response that passed the criterion and those which did not.
5. Display the results of the two groups for each criterion. To determine statistical significance of the results first display the two groups in the form of cumulative distributions. Then use a non-parametric statistic, such as the Kolomogrov two-sample test statistic, to determine if the distributions are distinct for each of the criteria. Determine if a common region exists such that all design criteria are satisfied simultaneously. If this is not the case, the designer has several options.

First, he can relax one or more of the criteria and regroup the data. Since no simulation is involved (only a regrouping of results already obtained) the computation time required is negligible. Step 5 is then repeated.

Second, if the design criteria cannot be relaxed he can obtain a more dense sampling of the parameter space likely to lead to satisfactory response. This is accomplished by starting again at Step 1 above, but with greatly reduced ranges on the parameter distributions. Information from the primary analysis is used to focus attention only in the regions where design satisfaction seems likely.

Finally, the possibility exists that the structure of the controller used to control the system is not capable of providing desirable system response. In this case another controller must be chosen and the design process repeated from Step 1. A selection procedure of controller structures will not be considered in this paper.

As an illustration of the above method, two examples are given. The first is strictly a linear control problem for which the results can be compared with those obtained by a classical approach. The second is a nonlinear problem. In this case the results are compared with the results obtained from linearizing the system using the Describing Function Method.

Example 1. PI Position Control with Delay

We wish to design a Proportional plus Integral (PI) controller to control a second order system with a delay. A schematic of the system is shown in Figure 1.

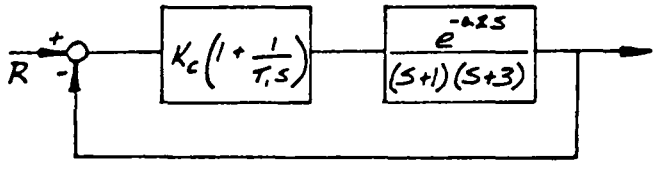


Fig 1 Schematic of Control System

From initial quiescence and for a unit step input, R, the design criteria are as follows: (1) A maximum of 20% overshoot will be tolerated. (2) The controlled position will be within $\pm$ 5% of the desired position for all time greater than or equal to 3 seconds (the settling time criterion). (3) The integral of the error squared, $\int (R-y)^2\,dt$, will be less than or equal to 0.75 (distance)^2 seconds.

The controller gains k_c and T_1 are the only uncertain parameters of this system Thus, our task is to select K_c and T_1 such that the above design criteria are simultaneously met (if this is possible) The a-priori distributions on the uncertain parameters were taken as uniform, k_c from 0 to 20 and T_1 from 0.1 to 5. The above system was simulated 500 times on a PDP-11/60 minicomputer The control gains were obtained each 'run' by randomly selecting values from the corresponding distributions After each simulation, the values of the design parameters and the gains which produced those values were recorded After the 500 simulations, for each design criterion, the values of the gains were separated into those which produced a response which passed the criterion and those which did not Density graphs of the gain-pairs which produced 'passes' are shown in Figures 2-4

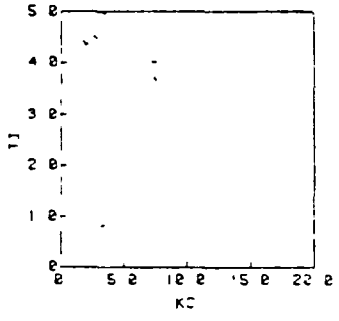


Fig 2 Overshoot criterion Pass, test point = 20%

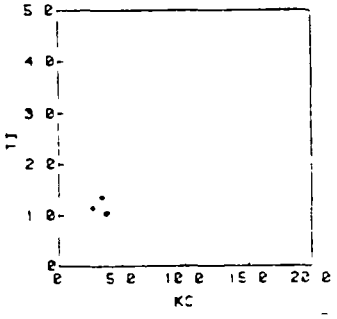


Fig 3 Settling time Pass, test point= 3 0 sec

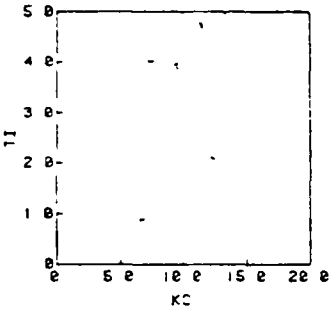


Fig 4 Integral Criterion Pass; Test point = 0 75

The computer time necessary to perform the simulations, regroup the data, and obtain the above graphs was approximately one hour As can be seen in the above graphs, a small region in the K_c, T_1 plane is common to all design criteria. Thus, if we choose a K_c, T_1-pair from this region of the parameter space our design task is complete To better determine this common region (since we only have four passes from the settling time criterion) this process could be repeated but with the range of the a-priori distributions on k_c and T_1 greatly reduced

A conventional method such as the Root Locus technique might be used to try to solve this problem. For $T_1 = 1$ we have pole-zero cancellation and the root locus for the system is shown in Figure 5

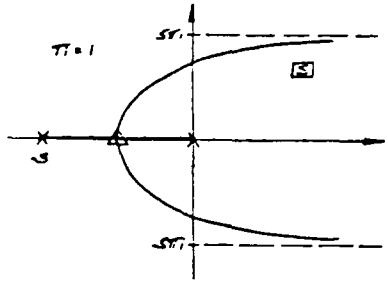


Fig 5 Root Locus for System of Fig 1 with $T_1 = 1$.

It appears that the optimal pole-placement is as indicated by the triangle (Figure 5) For this case, $k_c = 2\ 25$ However, while these gains produce a response which meets the first two criteria, the third one is not met From Figures 2-4 we see that $T_1 = 1$ and $K_c = 4$ will satisfy all design criteria

If $T_1 \neq 1$ (take $T_1 = 1.05$) pole-zero cancellation no longer exists and the root locus is drastically altered From the resulting root locus, it is uncertain that any value of k_c will satisfy the settling time criterion as the slowest eigenvalue appears to be slower than that required to produce the desired response As stated earlier, the problem stems from the fact that classical methods of controller design provide indirect results. To obtain the information specified by the design criteria listed above from these methods we must solve for the motion For any but the most trivial examples, this is impractical.

Example 2 Response of a Nonlinear Syster to a Disturbance

It is desired that the nonlinear system shown in Figure 6 respond to a unit step disturbance such that the following criteria are met (1) A maximum of 20% overshoot will be tolerated (2) The output variable, y, will be within $\pm$ 10% of the unit disturbance for all time greater than or equal to 5 seconds (3) The absolute value of the steady state error must be less than or equal to 5% of the disturbance Our task is to choose k such that these criteria are satisfied

The sinusoidal-input Describing Function Method will be used to help determine the range of k. The negative inverse of the describing function, N(a), for the relay with dead zone and the transfer function for the linear blocks, G(jω), were plotted in the Nyquist plane To avoid a limit cycle k must be chosen such that 0 465 k < $\frac{\pi\Delta}{2M}$ or k < 0 676 (Δ and M are shown in Figure 6)

However, this method does not help us determine if the above criteria can be met or what value of $k < 0.676$ we should use if they can be met. In addition, the describing function method leads us to believe that if $k < 0.676$ the system will be asymptotically stable. This is definitely not the case.

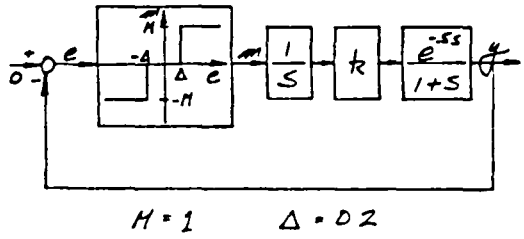


Fig. 6 Schematic of Nonlinear Control System.

Using the simulation-based approach the above system was simulated 200 times obtaining each run by randomly selecting a value for k from the assumed uniform distribution. For each criterion the gains were then grouped into those which did and did not produce a response which satisfied the criterion. The gains which produced 'passes' are shown in Figures 7-9 on the horizontal axis versus the frequency of occurrence in an interval of range 0.01.

To satisfy the design criteria simultaneously it is seen that k must be chosen close to zero. Thus, the design is complete.

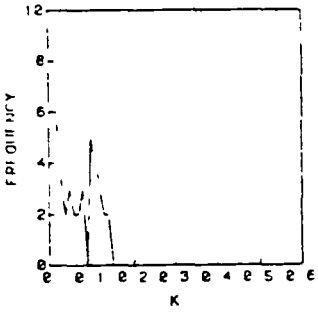


Fig 7 Overshoot criterion Pass, Test point = 20%.

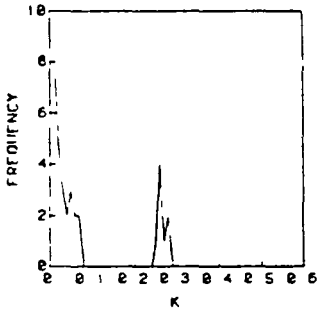


Fig 8 Settling Time Pass, Test Point= 5.0 sec

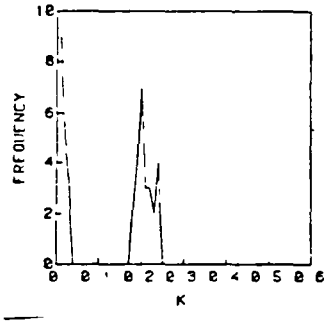


Fig. 9 Steady State Error Pass, Test Point = 0.05

METHOD FOR UNCERTAIN PLANT PARAMETERS

When plant parameters are uncertain, in general, we are no longer 100% confident that a specified set of controller gains will or will not lead to satisfaction of design criteria. In this case, it is the job of the designer to maximize the probability of success given the uncertainty in the plant parameters. Here, uncertain plant parameters pertains to uncertainty in the input variables and in the initial state as well as in the system parameters.

The procedure is as follows:

1) Assign probability distributions to all of the unknown plant parameters as well as control gains. Again, if there is no information to indicate otherwise, take these distributions to be uniform.
2) Randomly select a set of plant parameters and controller gains form the a-priori distributions assigned in Step 1 and simulate the system. Record values of the variables and the subsequent measure (or measures) of the system response at the end of each 'run.'
3) Repeat Step 2 to obtain a significant number of trials. Here, use the Kolmogorov statistic [2] and the desired level of confidence to help determine the minimum number of simulations that must be performed.
4) After the simulation runs are complete, for each criterion, separate the parameters into two groups, those which produced a system response that passed the criterion and those which did not.
5) Display the results of the two groups for each criterion in the form of cumulative distributions and use nonparametric statistics (such as the Kolmogorov two-sample test statistic) to determine if these distributions are distinct. Look for regions in which there is a high probability of 'passes' (or a low probability of 'fails').

If such regions exist, choose the controller gains to maximize the estimate of the probability of a satisfactory system response. This may not be possible at this point so the designer has several options.

First, he can relax one or more of the criteria and regroup the data. As stated previously, no simulation is involved and the computation time to perform this step is negligible. Step 5 is then repeated.

Second, if the design criteria cannot be relaxed this may indicate that one or more of the plant parameters must be known to a smaller range of uncertainty to be able to specify a design. The cumulative distributions are used to determine which plant parameters are sensitive to satisfaction of the design criteria.

Finally, the possibility exists that the structure of the controller used to control the system is not capable of providing desirable system response. In this case another controller must be chosen and the design process repeated from Step 1.

As an illustration of the above method, we present the following example.

Example 3 PID Control of a Continuous-Stirred Tank Reactor

It is desired that a PID controller be used to control a continuous-stirred tank

reactor (CSTR) about its unstable equilibrium point, T = 400°K, C = 0.5 g-mole/ℓ, where these symbols are defined below. The reactor is operating at nominal conditions when there is a sudden decrease in temperature in the reactor of 5 degrees Kelvin. The design criterion for the controlled system is as follows: (1) A maximum of 5°K overshoot will be tolerated. (2) The controlled temperature will be within ± 2°K of the desired temperature for all time greater than or equal to 1 minute. (3) The number of set point crossings shall not exceed 2.

The dynamic equations given in [4] and [5] are

$$V \frac{dC}{dt} = q(C_o - C) - VR$$

$$V C_p \frac{dT}{dt} = q C_p (T_o - T) + \Delta H \, VR - U(T - T_A)$$

where

$$R = k_o C \exp(-Q/T)$$

and where

- C = concentration of the reactant of interest
- C_o = feed concentration of the reactant of interest
- C_p = heat capacity per unit volume of the flowing materials as well as the contents of the reactor
- ΔH = molar heat of reaction, assumed exothermic
- k_o = frequency factor
- q = influent flow rate
- Q = ratio of the Arrhenius activation energy for the reactant to the gas constant
- T = absolute temperature of the reaction mixture
- T_A = average coolant temperature
- T_o = feed temperature
- U = overall heat transfer coefficient, in units which include the area of the transfer surface
- V = constant reaction volume

The heat removal term involving $(T-T_A)$ is modeled as is given in [6]

$$(T - T_A) = \frac{T - T_w}{1 + 1/F}$$

$$F = \frac{2 Q_c C_{pc}}{U}$$

$$U = C_1 Q_c^{1/3}$$

where

- C_{pc} = heat capacity per unit volume of the coolant
- C_1 = constant
- Q_c = volumetric flow rate of the coolant
- T_w = inlet coolant temperature

The relationship between the flow rate of the coolant and the controller output is shown in Figure 10. The fraction controller output is given by

$$C = \frac{G - G_{min}}{G_{max} - G_{min}}$$

where $G = C_2 Y + C_3$ (C_2, C_3 constants). Y is the output of the PID controller whose transfer function is

$$\frac{K_c}{1 + \tau s} \left[1 + \frac{1}{T_1 s} + T_d s \right]$$

which is given in [7]

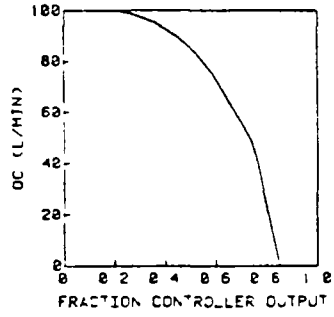


Fig 10 Control Valve Characteristic

Numerical values (nominal) for the reactor parameters were taken from [5] and [8]. A few of the constants associated with the controller were taken from [6]. These values are as follows

$C_o = 1 \frac{g\text{-mole}}{\ell}$ $\quad C_2 = 0.1068 \frac{ma}{°K}$

$C_p = 100 \frac{cal}{\ell - °K}$ $\quad C_3 = 15.76$ ma

$C_{pc} = 1000 \frac{cal}{\ell - °K}$ $\quad G_{max} = 20$ ma

$C_1 = 1357 \frac{cal/(min - °K)}{(\ell/min)^{1/3}}$ $\quad G_{min} = 4$ ma

$\frac{\Delta H}{C_p} = 200 \frac{°K}{(g\text{-mole}/\ell)}$ $\quad T_o = 350$ °K

$k_o = \exp(25)$ min^{-1} $\quad T_w = 350$ °K

$\frac{Q_c}{V} = 1$ min^{-1} $\quad \tau = 0.05$

$Q = 10^4$ °K $\quad V = 50$ ℓ

The two parameters associated with the reaction rate, R, will be treated as uncertain. The frequency factor, k_o, is assumed to be within ± 5% of its nominal value given above while Q, the ratio of the Arrhenius activation

energy to the gas constant, is assumed to be within ± 0.6% of its nominal value. Uniform probability distributions will be assigned to both of these parameters.

Before examining this case, however, it is interesting to consider the case where the plant parameters are fixed at their nominal values. Here, only the control gains K_c, T_i, and T_d capable of producing a desirable system response are unknown.

For reasons of economy only selected results obtained for the second design criterion (settling time criterion) will be presented. The procedure is identical for all other design criteria.

The above system was simulated 200 times, randomly selecting a 'set' of control gains from a-priori uniform distributions for each run. The gains were then separated into two groups, one which produced a response which passed the settling time criterion and one which did not. The cumulative distributions of the two groups for each gain are shown in Figures 11-13. Using the Kolmogorov two sample test statistic it appears that at the 99% confidence level the 'pass' and 'fail' distributions separate only for the gain K_c. However, there is a region about $T_d = 5$ where no 'fails' occurred. Separating the control actions, i.e., defining $T_d' = K_c T_d$ and $T_i' = K_c/T_i$ it is clear that these distributions do indeed separate and that a design exists such that the settling time criterion can be satisfied. If we obtain the probability distribution (more accurately, the frequency distribution) from the cumulative distribution for T_d' we see in Figure 14 for example that if K_c and T_d are chosen such that $44 \leq T_d' \leq 56$, the estimate of the probability is 1 that the settling time criterion will be met (23 passes out of 23 samples). Using the confidence limits for the binomial distribution [9] we can say that at the 95% confidence level if p = probability of passing the settling time criterion, $0.84 < p \leq 1$.

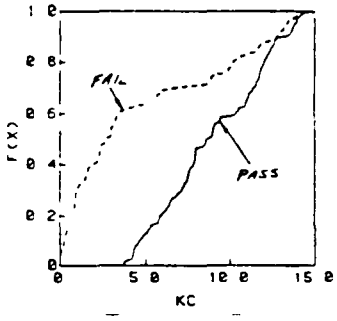


Fig. 11 Cummulative Distributions for K_c, Settling time

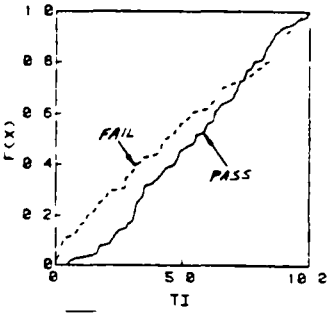


Fig 12 Cumulative Distributions for T_i, Settling time

If $34 \leq T_d' \leq 70$ the estimate of the probability is 55/57 ≈ 0.965 of success since in this region there are only 2 fails and 55 passes. Again, using the confidence limits for the binomial distribution, at the 95% confidence level we can say that $0.93 < p < 0.99$.

If it is desired to obtain a better determination of the confidence limits, T_d' should be restricted to the desired region (say $44 \leq T_d' \leq 56$) and more samples should be obtained from this restricted region.

Assuming that the original sample size of 23 is increased to 100 and still no fails occurred, we would be 95% confident that $0.96 < p \leq 1$.

Now we consider the plant parameters k_o and Q to be uncertain to the degree stated above. This system was simulated 400 times, again separating the parameters into two groups according to the binary design criterion. The 'pass' and 'fail' cumulative distributions for T_d is shown in Figure 15.

Although the distributions for the gain T_d do separate at the 99% confidence level it is not apparent what value of T_d should be chosen such that the settling time criterion would most likely be satisfied. In fact, when the control actions are separated the 'pass' and 'fail' distributions for T_d' do not separate. Obtaining the 'probability' distribution from the cummulative distribution for T_d' (Figure 16) we see that there is no region in which the probability of success seems likely.

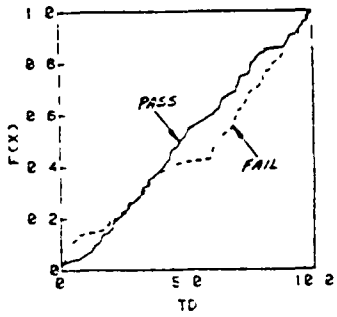


Fig 13 Cumulative Distributions for T_d, Settling time.

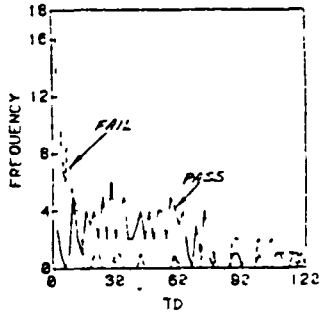


Fig. 14 'Probability' Distributions for T_d', Settling time

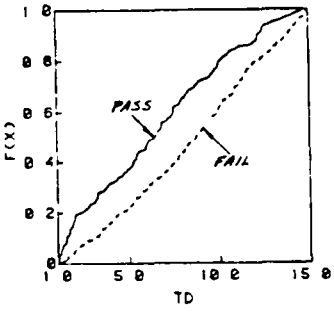


Fig 15 Cumulative Distributions for for T_d, Settling time

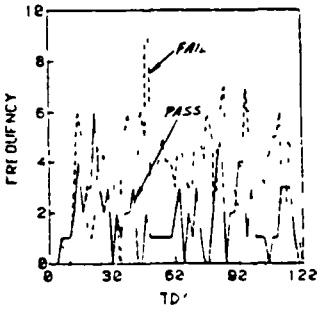


Fig. 16 'Probability' Distributions for T_d', Settling time.

The cumulative distributions for k_o and T_1 indicate that passing of the settling time criterion is rather insensitive to these parameters for the specified range of uncertainty In fact, a correlation analysis between each of the variables shows that the only 'significant' coefficient ($r = -0.525$) is between T_d and Q for the 'pass' distribution

It should be noted that if the plant parameters are fixed anywhere in the region of uncertainty a design exists for the settling time criterion however, when k_o and Q are unknown to $\pm$ 5% and $\pm$ 0.6% respectively, a design is not apparent One way of interpreting this is that in order to design the PID controller Q must be known to a lesser degree of uncertainty This would indicate that experimentation should be done to help determine Q more accurately.

It should also be noted that if one were to attempt a design using 'worst case' analysis that this scheme would fail For the example given above, the worst case is the combination of k_o and Q such that the reaction rate, R, is either a maximum or a minimum for a given state of the system

If k_o and Q are chosen within their respective ranges of uncertainty such that R is a maximum for given C and T, as stated previously, a controller design exists such that the settling time criterion is met However, if k_o and Q combine such that the minimum reaction rate is achieved, this design will fail the settling time criterion

Also, if k_c, T_i and T_d are chosen for the minimum reaction rate such that the design criterion is met, and k_o and Q combine to produce the maximum reaction rate, the system will fail the settling time criterion

CONCLUSIONS

The proposed design method helps us solve the following problem Given a dynamic system (generally nonlinear) with uncertain system parameters, initial state, input variables, or any combination of these, locate regions in the parameter space which lead to a desirable system response We have seen that this simulation-based method utilizes a generalized sensitivity approach which separates the parameter space into regions which produce a system response that satisfies given design criteria and regions which do not

The method provides direct results; i e for a fixed set of parameters satisfaction or nonsatisfaction of the criteria is determined However, for systems requiring 'large' amounts of time for this determination the proposed method may not be practical

Finally, it was seen that in some cases a parameter may have to be known to a lesser degree of uncertainty if a satisfactory design is to be specified This parameter (or parameters) is identified using the generalized sensitivity analysis incorporated into the design method

ACKNOWLEDGEMENT

This research was supported in part by NASA cooperative agreement number NCC 2-67 from the NASA Ames Research Center

REFERENCES

1. Tiwari, J and Hobbie, J , "Random Differential Equations as Models of Ecosystems Monte Carlo Simulation Approach," Mathematical Biosciences, Vol 28, 1976, pp. 25-44

2 Spear, R. C , "The Application of Kolmogorov-Rényi Statistics to Problems of Parameter Uncertainty in Systems Design," International Journal of Control, Vol. 11, No 5, 1970, pp 771-778

3. Spear, R. C. and Hornberger, G M , "Eutrophication in Peel Inlet-II Identification of Critical Uncertainties Via Generalized Sensitivity Analysis," Water Research, Vol 14, 1980, pp 43-49.

4 Bilous, O , and Amundson, N , "Chemical Reactor Stability and Sensitivity," A. I Ch E Journal, Vol 1, No 4, 1955, pp 513-521

5 Perlmutter, D., Stability of Chemical Reactors, Prentice-Hall, Inc , New Jersey, 1972, pp 6, 7, 33.

6 Ramirez, W. and Turner, B , "The Dynamic Modeling, Stability and Control of a Continuous Stirred Tank Chemical Reactor," A I Ch E Journal, Vol 15, No 6, 1969, pp 853-860

7 Takahashi, Rabins, and Auslander, Control and Dynamic Systems, Addison-Wesley Publishing Company, 1970, p 195

8 Aris, R and Amundson, N., "An Analysis of Chemical Reactor Stability and Control - II," Chemical Engineering Science, Vol. 7, 1958, pp 132-147

9 Clopper, C J and Pearson, E S , "The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial," Biometrika, Vol 26, 1934, pp 404-413

PARASOL-II: A Laboratory Simulation and Control Tool for Small Computers

David M. Auslander
Mechanical Engineering Department
University of California, Berkeley

Abstract

An interactive computing language designed for dynamic system simulation, report-quality graphics, data acquisition, and simple, real-time control is described. Its major features include completely general equation definition, including user-defined functions and imbedded algorithmic code, sorting of algebraic equations, and totally user-definable commands. The underlying interpreter is written entirely in the compiled language C. It is compact enough to run on microcomputers and is portable to several popular mini or microcomputers. A command set for simulation of differential equations and mixed differential and difference equations is described, as is another command set for graphics. Sample problems are solved using the simulation and graphics capabilities.

Introduction

The computing environment described in this paper comes from a history of providing computing support for a university laboratory involved in teaching and research in control and dynamic systems. The needs of this lab have included:

- -dynamic system simulation; ordinary differential equations, difference equations, and systems which combine both differential and difference equations.
- -graphics; primarily report quality graphs of analytically or experimentally obtained data.
- -control; real-time control of a variety of process and servo systems.
- -data acquisition; recording of experimental data for later analysis.
- -microprocessor development; teaching of microprocessor use in engineering systems and development of microprocessor-based systems.

The philosophies guiding this development have been, for hardware, that single-user computer systems can meet all of these needs and are most suitable for those tasks involving real-time applications. For software, the transient nature of the user population demands easy-to-use and easy-to-learn systems, but, at the same time, the diversity of jobs to be done, even within a single task area, requires the flexibility of programmable systems.

The choice of single-user computers is based mostly on considerations of speed and ease of access to I/O facilities for real-time tasks, extensive use of graphics with its high data-rate requirements, software simplicity, and the modest size of most of the computing tasks undertaken. The size of a single-user computer system is primarily limited by the economics of how much computing resource can be reserved for the exclusive use of a single individual for long time periods (perhaps permanently). In the current marketplace (1982), the economics plus performance constraints put the target computers for this work into the $2,000 to $20,000 range (computer, memory, terminal, mass storage; not including additional peripherals such as printers, graphic devices, A/D, D/A ,etc.)

For the last several years, our software needs have been met by a simulation language (Parasol[1]), a graphics/data acquisition program, and various special purpose programs. The motivations for undertaking the development of a new computing language were the desires to: 1) increase portability of the software to a variety of computers, 2) improve programmability, 3) introduce programmability at the command level, and, 4) combine several software packages into one. Although these desires are not new, recent developments in both hardware and software have not only made the project possible, they have also made it necessary!

On the hardware side, the middle half of the price range cited above is rapidly becoming the prime territory for the 16-bit processors. The upper part has always been dominated by 16-bit CPU's, but now new processors are joining the old ones. On the other hand, there is no sign that 32-bit systems will move down into that range soon. Memory, both main memory and mass storage, is continuing the per-bit reduction in cost that has been going on since computers were first marketed.

On the software side, the development of small computer, high level languages with sufficient power and efficiency to be the vehicles for writing their own compilers has led to truly portable languages that can be marketed for many combinations of processor and operating system. In particular, the language C[2] combines language elements useful for small computer applications, recursive, potentially reentrant structure, and commercially available implementation for several computer systems. Having the same implementation, not just the same language, is a very important factor for portability. The same power that makes these languages' portability possible also means that our goals relating to increased programmability can be met more easily.

System Specifications

Portability, ease-of-use, ease-of-learning, programmability, and low development cost all point to an interpretive system. The compromise in using interpretive systems is in execution speed. Some of the loss in execution speed, however, can be recouped by using semi-compiled, threaded code base on reverse-Polish notation (RPN). Threaded code, in which all internal references are by address rather than be name, can be utilized most effectively if the

user-level language is also based on RPN. These specifications carry over from previous versions of Parasol. Several years of experience in using Parasol have shown that the inconveniences associated with using RPN were more than offset by added flexibility in function calling and the relatively compact, simple interpreter that resulted.

In addition to the desire for greater portability, there were several major extensions of Parasol that would extend its usefulness greatly; except for the limited inclusion of imbedded sequential code, it was never worthwhile considering any extension of further implementation of Parasol within its mixed Fortran/assembly language structure. These extensions include:

- -User-defined functions at the interpretive level; this is very important in generating compact, readable code.
- -Multiple, interacting simulation blocks; the most pressing application of this is for discrete-time control of continuous-time systems. It is also useful for multiple-time-scale problems.
- -Integration of full graphics support and simulation under the same software umbrella.
- -More flexible command structure.
- -Dynamic memory allocation.

A philosophy that is relevant to meeting these goals was discussed in the context of Xerox's research computer language, Smalltalk[3]. That discussion emphasized the concept of "mode-lessness." What this means is that tasks at different control levels are all defined and executed by a common mechanism. This avoids many of the usage pitfalls of current computing systems that require different conventions in different modes. For example, editing, compiling, and running a Fortran program all operate according to completely different sets of rules. At the run phase, in fact, each program usually has a completely different structure for interacting with the user. Looking at the wish-list for the new Parasol, and recalling that the RPN structure already existed and was to be retained, an easy step towards a more mode-less system could be made by adopting some of the syntactical conventions of Forth[4]. The basic idea, as it applies to a Parasol-like application, is that the RPN stack is accessible to the user via the console. The user can really do only two operations: enter values onto the stack or cause a function to be executed. That's all! This mechanism is used for all of the usual user/computer interaction -- data entry, program entry and modification, inspection of internal states, command entry, and command definition. In this manner, functions and operators used for computational purposes (+, -, sin, cos, etc.) are handled in exactly the same manner as what are usually referred to as commands (input, draw, set, run, etc.) This introduces a degree of mode-lessness into the system in the sense that the mechanism used for function definition is also used for command definition. As in Forth, this structure greatly simplifies debugging because any function or command can be exercised directly by the user by manually placing the arguments on the stack, invoking the function, then observing the results on the stack.

Function definition is provided through a define-function function It, however, requires that the user input be interpreted in a different manner than normal input, so use of that function switches the user to an input mode. Conditionals and loops are implemented within functions, but no specific syntactical structure is provided because they are also implemented with functions.

Providing for simulation of systems with interdependent, but mathematically distinct sections (such as a discrete-time controller and a continuous-time system) requires the definition of an entity that can be called a simulation block. As long as the syntax within a simulation block bears a strong resemblance to the previous Parasol syntax, users should have little difficulty shifting from old to new.

This is all that is required in the way of general specification for Parasol-II. The next two sections deal with the actual implementation, that is, the consequences of combining the wish-list with the general specification. The first of these sections gives a user's-eye view and the second examines some of the internal structure.

Parasol-II: User's View

Getting Started

Input to Parasol-II comes through an input stream; text output is sent to an output stream. These streams can be set to any file or device. In the standard implementation, the input stream is initially set to a specific file, which allows for start-up configuration. The output stream is initially set to the console device. On reaching the end of the start-up file, or a close-file, the input stream reverts to the user console ($icls; actual function names will be given in parentheses following their generic description -- the $ is part of the name).

To look at Parasol's most primitive capabilities, let's assume for the moment that the start-up file is empty. After getting the start-up message, the user will get the Parasol prompt, p2>. The user can then enter items. An item is a string of characters with no intervening blanks, tabs, or returns. Blanks, tabs, or returns serve to separate items from one another. At this stage, the item typed could be a numeric constant, a primitive (i.e., predefined) function name, or an item that defines a variable. If the item is a constant, its value will be put on the stack. If it is a function, the function will be executed. If it is anything else (or if the function cannot execute correctly), an error message will result. Variable-definition is a special case of putting a constant on the stack.

In most implementations of Parasol, the actual console interaction is handled by an operating system. Most operating systems pass input information to an executing program in complete lines. For that reason, a series of items will only be interpreted by Parasol in such a system after an end-of-line indication (usually a return) has been typed. To Parasol, however, there is no special significance to the unit represented by a line, so it makes no difference whether items are typed one per line, or several to a line. (The data-input mode, described below, is an exception to this rule.)

Defining Variables

Variables have much the same meaning in Parasol that they have in other computing languages, such as Fortran or Basic. Use of the variable name (up to five characters) as an item causes the value of the variable to be put on the stack. In addition to this use of a variable name, Parasol provides for the prefix modifier &, which stands for "address-of."

When used immediately preceding a variable name, with no intervening spaces, it causes the address identifying the variable, rather than its value, to be put on the stack. Using this prefix with a variable name that has not yet been defined causes Parasol to first define the variable (i.e., allocate storage) and then to put the address of the newly-defined variable on the stack.

This mechanism can be used to give initial values to variables by using the store ($sto) function. $sto first takes a variable's address off the stack, then takes the next value off the stack and stores it as the variable's value. Thus the sequence of items, 1 &x $sto will define the variable x, if necessary, and give it the value 1. The prefix modifier @ (for the mnemonic store-at) provides a shorthand notation for the store operation, so 1 @x is equivalent to the above sequence.

Use of the address-of prefix is not limited to variables. Any type of item can have its address placed on the stack. This feature will be very useful for a variety of operations.

Functions and Simulation Blocks

Define-function ($dffn) is the function used to input user-defined functions (note that it is also a function). The syntax is:

```
$dffn fname -body of function- $$
```

The body of the function is any sequence of items; fname, the function name, can be up to five characters. The $$ terminates the definition. Since Parasol pays no attention to ends-of-line (except as an item separator), the definition can be spread across any number of lines. While Parasol is in function-definition mode, the prompt dffn> replaces the usual p2> prompt if more than one line is used. Re-use of a define-function with a previously defined function name causes the new function to replace the the old one.

Define-simulation-block ($dfsb) provides for the input of a set of state equations. The syntax of the definition is:

```
$dfsb sbname -body of block- endsb
```

The body of the simulation block includes equations of two types: equations that define state variables (dynamic equations) and auxiliary (algebraic) equations. Although in a mathematical sense it is not strictly necessary to allow for the auxiliary equations, they are essential to the efficient simulation of almost any reasonably complex system.

The prefix modifier # is used to mark a state variable. Depending on the kind of dynamic equation being written, the symbol # can be interpreted as, for example, d/dt for differential equations or unit time advance for difference equations. As far as the definition of Parasol is concerned, there is no particular preferred interpretation of # except that it marks a special kind of variable. The syntax for definition of a state variable is:

```
#sv = -right-hand side of equation- $$
```

The right-hand side of the equation is any collection of items, including conditionals, loops, and store's, if desired; it can also extend across as many lines as desired. It must, however, leave a single value on the stack when it is done. The differential equation dx/dt= -x, for example, could be written as: #x = x $chs $$. The spaces surrounding the "=" are necessary, as are the spaces surrounding other items. $chs is the change-sign function. $$ is the terminator for each such state equation.

The syntax for definition of an auxiliary equation is similar:

```
av = - right-hand side of equation - $$
```

The differential equation used above, for example, could have been written in the form:

```
#x = w $$
w = x $chs $$
```

The state and auxiliary equations can be entered in any order. For the state equations, most processing algorithms (e.g., a Runge-Kutta integration algorithm) are independent of the order in which the state variables are defined. The auxiliary variables are sorted into their preferred computing order prior to simulation. If sorting is impossible because of circular dependencies (e.g., a=b and b=a) the warning "static loop" is issued and simulation proceeds.

If the define-simulation-block function is used for a simulation block that has already been defined, the action of any state or auxiliary variable equations entered is to append those equations to the list of equations previously defined, unless the particular variable on the lefthand side of the equation has been previously defined, in which case the new equation will replace the old one.

Initial conditions can be set for any state or auxiliary variable in much the same way as values for variables are stored, except that the initial-condition function ($ic) is used instead of store. Limiting (saturation) values can be set for any state variable by use of the saturation-limit function ($sat). It works by specifying the low limit, high limit and state variable address. For example, the state variable x1 could be limited to the range -1 to +1 by the following sequence: -1 1 &x1 $sat.

Use of Files

For all but the simplest problems, it is necessary to preserve the user's program on a file, using the operating system. This file can be produced either by starting with an editor to write the Parasol program, or writing the program into Parasol, and then using the listing functions ($lstf for functions and $lsts for simulation blocks) to save it. In either case, when that file is to be used, it is necessary to switch the input stream to the file. This is done by calling the function $ifil (for input-file). $ifil will then ask the user to enter a file name which it will then open as the new input stream.

When the input stream is switched to a file, all subsequent input is taken from that file. Thus the information in the file is treated in exactly the same way as input information that is typed directly at the console. To make up an input file, use the operating system's editing facility to create a file that contains the same information that you would type when using Parasol interactively. Any material from a semicolon to the end of the current line is

treated as a comment and ignored by Parasol (this is true for lines that are typed from the console also, but, in that case, it is not a very useful feature).

An input stream may open another input stream, to whatever level of nesting is provided in the version of Parasol you are running. When an end-of-file or $icls function is encountered, the current input stream device or file is closed and the input stream drops down one level. The bottom level is the console.

Most user applications of Parasol make use of a library of functions and commands that are tailored to a class of problems. These functions are written in the normal Parasol user language, but provide the ability to do common tasks easily. Currently defined libraries include one for simulation and one for graphics. Because these libraries are written in Parasol user language, they can be easily modified by users to fit special needs. To further simplify the use of these libraries, the standard version of Parasol starts operation with its input stream set to a file named startp.p. This file can be set to read in whatever library is needed, to customize Parasol for a specific application, or to make Parasol perform a particular function completely automatically, with no user interaction at all.

Output files are created in a similar manner. The output stream is initially set to the console terminal. A new output stream can be opened by using the $pfil command which will ask the user for the name of the file to be used for output. All subsequent output, except for error messages, will be sent to the new output device or file. The $pcls command is used to return the output stream to the console, and to assure that the output file is closed properly (note: if files are not closed, many operating systems will delete them when the program finishes!).

Simulation

The simulation library package provides all of the facilities necessary to simulate sets of linear or nonlinear ordinary differential equations, sets of linear or nonlinear difference equations, or systems with both differential and difference equations. The output from the simulation can be in graphic form (if a suitable graphic device is available), in tabular form, or both. If desired, the tabular output can be directed to a file or printer by using the $pfil command to redirect the output stream.

The differential equations to be solved must be in a simulation block named sys; the difference equations must be in a simulation block named cntrl The default definition of these blocks is for an empty block. If only one of them is used, the other (which is empty) will be ignored by Parasol.

The graphics variables and scales are set up with the four functions .hv, .v1, v2, and .v3, for the horizontal variable and up to three vertical variables. Each of these function puts three values on the stack when they are called the value of the variable to be plotted, its minimum value, and its maximum value (for scaling). If the function .hv is empty, no plotting is done For example, to plot time on the horizontal axis and a variable named x on the vertical axis, the set-up commands would be:

```
$dffn .hv .t 0 .tmax $$
$dffn .v1 x -1 1 $$
```

The variables .t and .tmax are defined within the simulation package and are the running time and the termination time for the simulation. A function called "time" is defined for in "run" as:

```
$dffn time .t 0 .tmax $$
```

for convenience in assigning time as the horizontal variable.

Tabular output is specified through a function called print and a variable named .dtp, which determines the time interval for printing to occur. The function print contains a list of the variables whose values are to be printed. For example, to print time, and the variables x and v every 0.4 time units, the set-up commands would be:

```
$dffn print .t x v $$
0.4 @.dtp
```

A value of zero for .dtp causes printing to occur every time step.

If both the continuous-time, differential equation block, sys, and the discrete-time, difference equation block, cntrl, are defined, execution of the discrete-time portion is governed by the variable .tsmp, the sampling interval. "cntrl" will be run whenever the elapsed time since the last time it was run is equal to or exceeds .tsmp. If the step size and the sample interval are not even multiples, the actual sample interval will contain some dither due to round-off but will be correct on the average.

To run the simulation, the step size and maximum time are put on the stack as arguments to the "run" command, for example,

```
0.1 10 run
```

The differential equation integration is currently carried out by a fourth-order, fixed step size Runge-Kutta integration algorithm. A summary of the simulation package is given in appendix IV. Several simulation examples are included in appendix III.

Graphics

The graphics package provides a set of commands and functions for the production of report-type graphs from data that are on files or computed in user-defined functions. Tick marks, axes, labels and titles can be set up for the graphs, and then plotted. Scaling is done between the plotter-oriented variables used by the primitive graphics function ($gmov) and values that are in problem-oriented units. Additional scaling can be also be done to accommodate unit changes between the data actually on the data file and the system of units used for the graph.

The general procedure for producing a graph from a data file is to load Parasol, load the graphics command package (either through the startp.p file or by using the $ifil directly), use the various set-up commands in the graphics package to specify the physical layout of the graph, locations of axes and ticks, and labels and titles. (These set-up commands all start with "st," as in sttck for set-ticks, stlab for set-labels, etc.

The particular graphic device to be used can be selected with the $gdev function (2 $gdev sets the graphics output to device 2). Any set-up that is peculiar to the device (and most have several

peculiarities) is done with the special-graphics input ($gspc; 1 $gspc will give a menu of possibilities for the currently specified device). Any file or device specifications necessary to connect the graphic device to the program can be done with the $gfil command, which will prompt the user for a file or device name (many graphics devices do not require this).

The data to be plotted is assumed to be organized by lines, each line containing a number of data values. Each point on the graph consists of one point from the line as the x-coordinate, and one point for the y-coordinate. Which of the data values is used for x and which for y is controlled by the x and y plot variables (.xvar and .yvar), which can be set to any desired correspondence.

With all of the set-up completed, the actual drawing of the graph can be done by invoking the commands to draw axes, ticks, etc., as needed, followed by a call to the command "graph" to draw the graph. ("dgrph" can be used to draw the graph with dashed lines or "pgrph" to plot only the data points with a marker.) Graph will prompt the user for the name of the file that contains the data.

Any of these functions can be accomplished interactively, by typing directly into Parasol, or from files by redirecting the input stream to a specified file. In many cases, it is desirable to put the set-up information and data to be plotted in the same file. This can be done by putting a blank line after the call to graph. This will cause graph to leave the input stream where it is.

The use of the graph plotting functions is illustrated by example 4 in appendix III, in which the data that was produced by the simulation of the bouncing ball is plotted. Both position and velocity vs time are plotted as is a phase-plane plot of velocity vs position. A summary of the graph operations is given in appendix V.

Parasol-II: Inside View

Parasol-II is written entirely in C, using the C-compiler produced by Whitesmiths, Ltd. The major reasons for choosing the language/compiler combination are:
1) the language supports the data structures and operators that make writing an interpreter of this sort reasonably efficient,
2) the language is recursive, which is critical in the evaluation of user-written functions that access other user-written functions,
3) Whitesmiths' compiler produces re-entrant code so that real-time parts of the system that utilize interrupts can be written in C also,
4) Whitesmiths supports several popular processors (Parasol is currently running on PDP-11's (RT-11 and RSX-11), 8080/8085/Z-80 (CP/M or slave), 68000 (running as a slave to RT-11); VAX/VMS is also supported but Parasol has not yet been compiled for VAX),
5) separate compilation of program modules is part of the definition of the language,
6) the data types normally needed for our applications are supported (integer, long-integer, floating point, and double precision floating),
7) the generated code is reasonably efficient so the use of an interpreter can be considered,
8) dynamic memory allocation is supported.

There are also some disadvantages. C is not commonly available on popular mainframe computers (IBM, CDC, etc.). The Whitesmiths C library is not compatible with the UNIX C library; conversion of Parasol to UNIX is currently underway.

The concept of an item is central to the function of Parasol. The central programming construct is a data structure that is used to describe the features of an item. The types of items currently defined are: predefined (primitive) functions/operators, user-defined functions/operators, state variables, auxiliary variables, value variables, constants, simulation blocks, text blocks, link blocks (used internally to link non-contiguous sections of the item directory), temporarily reserved items, and undefined items. As each item is defined, an entry is created for it in the master directory. That entry has five parts, some of which have fixed usage, and others for which the usage depends on the type of item being defined. The first two parts are item-type and the item's name. There is a "link" portion that contains a pointer linking the item being defined to other items of similar type. The remaining two parts are pointers that vary with the type of item. For example, for state variables, one pointer is to the block where its numerical value information is stored. This block contains the current value, initial value, right-hand-side value (for use by simulation algorithms), and saturation limits. The other pointer links the state-variable item to the code for its right-hand-side expression.

All of the C code that represents the total Parasol processor revolves around either the master directory or the stack. The primitive functions, which are all written in C, have their primary connection with the rest of Parasol by getting values from the stack or putting results on the stack. Some of these functions also use the master directory since such operations as function definition and function listing are themselves primitive functions.

The basic structure of Parasol can be seen best by following the sequence of events that start when it is loaded. The key to all processing is the input stream. At the highest level, the sequence of events is simply get an item from the input stream then process it. If the item is a constant, put its value on the stack. If the item is a primitive function, call the corresponding C function. If the item is a user-defined function, call the function-interpreter. If the function interpreter finds another user-defined function, it calls itself. During program execution, this sequence constantly repeats.

The only deviation from that sequence is when the function being executed diverts the input stream. This can happen (currently) under three circumstances: 1) to compile the equations for a user-defined function or simulation block, 2) to treat the input stream as a data file in which case end-of-line is treated as a significant event, or 3) to read and store text strings which are, in effect, items that are delimited by ends-of-line.

The data input mode is the mainstay of the graph processor because data for graphs is located according to its position on the line (using the variables .xvar and .yvar). Text string definition is also used in the graph processor to allow the user to enter graph labels and titles. It is also used internally to print prompting messages to the user under various circumstances. Other possible uses include construction of formatted screens, putting

identifying information into data files produced with $pfil, etc.

Each of these input-stream-diverting modes is identified to the user by special prompts so there is no ambiguity as to the type of input that Parasol expects or what is necessary to get out of that mode.

Graphics, which plays such an important role in almost every Parasol application, is normally implemented through primitive functions. These functions only need to be capable of taking coordinate pairs and a "pen" value from the stack and drawing straight lines between points, and, if the device has character capability, taking a string of characters and printing them on the graphics device at a specified location. Because the demand on the graphics drivers is so minimal, it is very easy to add new graphic devices to Parasol.

Summary

The most important question to ask as a summary to a description of a programming language is, "what makes it unique?" Parasol combines features from many different sources, but appears to have a niche of its own. There are three types of computing languages that it must be compared to in order to see what it has drawn from each, and where it is different from any of them:

-Interpretive languages (Basic, Forth): Like these languages, Parasol is interactive. Unlike them, it contains internal structures to define state and auxiliary variables and simulation blocks, as well as integration algorithms. Unlike Forth, which also allows user definition of command structures, Parasol is designed for a computational environment and, although not strictly limited to those tasks, is specifically designed for simulation, graphics, data acquisition, and machine or process control.

-Compiler languages (Fortran, C, Pascal, etc.): These languages are not interactive. Other than that, the above comments apply.

-Simulation languages (CSMP, CSSL, ACSL, etc.): These languages contain the same simulation constructs as Parasol (state variables, etc.). Many of them are not interactive (they take the simulation input and translate it into another language, such as Fortran, which must then be compiled). Most are large enough to require a very large mini computer (VAX size) or a mainframe to run. Those that don't, often have more limited abilities. Parasol is compact enough to run in a microcomputer without sacrificing the functional capabilities of the larger simulation languages. It also allows the most flexibility at the command level, because commands are totally user definable.

References

1. Auslander, D. M., "A Continuous-Systems Simulation Language Designed for LSI Economics," Mathematics and Computers in Simulation, XX, 308-313, 1978.

2. Kernighan, B. W., and D. M. Ritchie, The C Programming Language, Prentice-Hall, Inc., 1978.

3. Smalltalk is featured in a series of articles in Byte, 6, 8, McGraw-Hill, August, 1981.

4. Forth is described in a series of articles in Byte, 57, 8, McGraw-Hill, August, 1980.

APPENDICES

Because of space limitations, only Appendix III, examples, is included in this version of the paper.

APPENDIX III. Parasol-II Examples

Example 1. Two-Dimensional Positioning Table

We are interested in simulating and controlling the motion of a two-dimensional positioning table. For this example problem, we will assume that the table obeys simple, linear differential equations of motion based on a force input from a motor on each axis and linear damping. With x1 and v1 representing position in the x and y directions, and v1 and v2 representing the velocities, we can write the differential equations as:

```
(d/dt)x1 = v1
(d/dt)v1 = (f1 - b1 * v1)/m1
```

and

```
(d/dt)x2 = v2
(d/dt)v2 = (f2 - b2 * v2)/m2
```

where f1 and f2 are the force inputs, which are proportional to the inputs to the motor controllers, u1 and u2, b1 and b2 are the damping constants, and m1 and m2 are the table masses. If we assume that the table is constructed so that the x-axis motor and ways ride on top of the y-axis structure, the mass of the y-axis will have to include the x-axis.

In this, and in example 3 where we apply discrete-time velocity control to this system, we will assume that we are trying to make the system move through the motion shown in Figure ex1.1. In this example, we will have to provide a table of values for the inputs, u1 and u2.

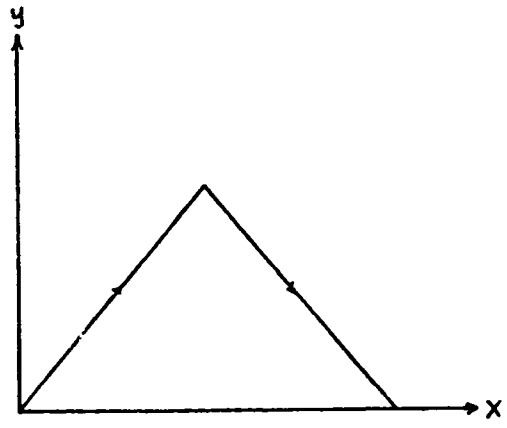


Figure ex1.1 Desired Path

The program listing is shown in Figure ex1.2. The listing was created as a separate file using an editor, and will have to be read into Parasol to run the simulation. Each of the input lines are identical to the lines that would be typed to enter the equations interactively except that Parasol would provide appropriate prompts which do not appear on this listing. The comments could be typed interactively also, but would not be stored anywhere.

```
;Two-Dimensional Positioning Table
;Parasol-II sample problem. This sample illustrates
;straightforward solution of sets of differential equations.
;Although these equations are linear, the format and
;procedures for solving nonlinear problems are identical.

$dfsb sys
#x1 = v1 $$                ;define the position state variable for axis #1
#v1 = fm1 b1 v1 * - m1 / $$      ;velocity state equation
fm1 = km1 u1 * $$                ;force applied to table by drive motor --
                                 ; u1 is the controlling input.
;Same equations for axis #2 --
#x2 = v2 $$                ;define the position state variable for axis #2
#v2 = fm2 b2 v2 * - m2 / $$      ;velocity state equation
fm2 = km2 u2 * $$                ;force applied to table by drive motor --
                                 ; u2 is the controlling input.
;The controlling inputs, u1 and u2, will be defined by tables. The
; shape we are trying to draw is a corner, which requires a constant
; velocity in the x-direction and a y-velocity that reverses at
; the corner.
;The ability to spread equations over many lines
; will be used in this case to increase the readability of the tables.

u1 =                             ;Input table for x-input -- constant until the
        0       1                ; time to stop the motion, then slow down to
        9       1                ; zero input.
        10      0
.t 3 $fng $$                     ;Complete the definition with independent
                                 ;variable (.t), number of pairs (3) and the
                                 ;call to the function generator ($fng).

u2 =                             ;Y-input table. The first entry is time, the second
        0       1                ; is the value. Spreading an equation across several
        4.9     1                ; lines is used in this case to increase readability.
        5.1     -5               ;4.9 to 5.1 is the time to reverse the velocity.
        6       -1               ;Use large braking force, then tc -1
        9       -1               ;Beginning of stop region.
        10      0
.t 6 $fng $$                     ;Complete the definition with the independent
                                 ;variable (.t), the number of pairs (6), and
                                 ;the call to the function generator ($fng).
endsb                ;end of simulation block

;Set values for constants (these values can be changed while running the
; problem by typing similar lines with new values).
1 @m1                ;mass of table #1
0.2 @b1              ;damping coefficient -- includes friction and damping
                     ; effects caused by coupling back through the motor.
1 @km1               ;force produced by motor #1
2.5 @m2
0.2 @b2
1.5 @km2

;Define variables for plotting -- plot x1 vs x2 to get a picture of
;the path. A subsequent run will be used to get the velocity vs time
;curves (or that data could be sent to a file for later plotting).

$dffn .hv x1 0 50 $$       ;horizontal variable and scale
$dffn .v1 x2 0 15 $$       ;vertical variable #1 (the only one in this case)

;Once this file is read in, the user can initialize the graphics device and
; give the run command
```

Figure ex1.2 Parasol Program Listing

The program starts with the definition of the simulation block. The standard simulation package, "run," is set up so that the differential equations to be solved will be in a simulation block named "sys." The equations themselves are transcribed directly from the equations listed above, with the suitable translation into reverse Polish notation. Following the simulation block definition, the function $fng is used to define tables for u1 and u2. It takes paired inputs and gives either an interpolated or non-interpolated output based on the value of the specified independent variable, which is ".t" in this case for running time (.t is a reserved name used by "run" for time).

Values for the constants follow. These are all active statements that cause the indicated value to be stored at the named variable specified. The symbol "@" is read, "store-at." Finally, the specification for the graphic output is given. Graphic output is defined by the functions .hv (for horizontal variable) and .v1, .v2, and .v3 for up to three vertical variables. In this case, the horizontal variable is specified as x1, the x-axis position, and the vertical variable is x2, the y-axis position, so the actual path taken by the table will be drawn out.

The actual console interaction (on a CP/M system) necessary to run this system is given in Figure ex1.3. The graphics device in that case was a video display system. Figure ex1.4 shows the output redrawn using a pen plotter. Note that the graph does not contain any tick marks, labels, etc. The normal "run" package does not contain functions to do that in order to conserve on memory usage. Un-notated graphs are usually adequate for design and debugging use. When notated graphs are needed, as for reports, the usual procedure is to use the "print" and "$pfil" functions to produce a data file with the results and then to use Parasol's "graph" package to make an annotated graph. This is done in example 4, where the data produced from example 2, the bouncing ball, is drawn (see Figure ex4.2).

```
A>par2
Parasol-II; last updated 24-Feb-82

Input stream is initially set to file: startp.p
p2>$ifil        ;read in the equation file
File: 2d.p
p2>$gini        ;initialize the graphics
Godbout Spectrun Graphic board
p2>axes
p2>.1 10 run    ;run the simulation, dt=0.1, tmax=10
p2>
```

Figure ex1.3 Console Interaction for Two-Dimensional Motion Simulation

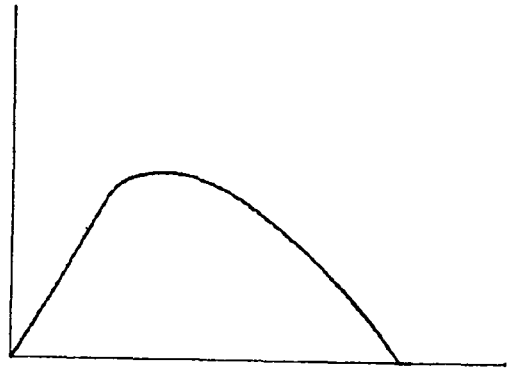


Figure ex1.4 Simulated Response of Two-Dimensional Motion

Example 2. Bouncing Ball

The problem of simulating a mechanical system including some sort of bounce, that is, intermittent contact, requires the use of decision logic because the effective differential equation changes depending on whether the object is in contact or not. A simple example of this is the bouncing ball. In this example, we handle the bouncing ball problem by assuming that there is a very stiff spring located near the ground, Figure ex2.1. The differential equation describing this motion is:

$$(d/dt)x = v$$
$$(d/dt)v = [fs + fd - fg]/m$$

where

$$fd = -b\,v \qquad \text{(damping force)}$$
$$fg = \text{const} \qquad \text{(gravity force)}$$

and

$$fs = \begin{cases} 0 & \text{if } x > xz \\ k\,(xz - x) & \text{otherwise} \end{cases}$$

The Parasol program to implement this, Figure ex2.2, uses the if-then-else construct to simulate the intermittent contact. This is accomplished with the Parasol functions $iftr, $else, $ndif.

Although this program will give a correct result, it is highly inefficient because the very stiff spring demands a very small step size in the integration algorithm in order to meet both accuracy and stability constraints. In this problem, however, that can be overcome without using either a variable-time-step integration algorithm, or a stiff equation algorithm. The step size can be made a function of position and set to a small value when the ball gets close to the spring. When it moves away, the larger step size of free flight can be used. This is done in the program by setting the step size, which is the reserved variable .dt, equal to the desired step size, depending on x.

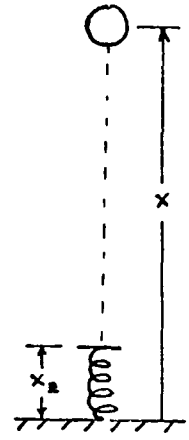


Figure ex2.1 A Model of the Bouncing Ball Problem

The definition of the simulation equations ends with the "endsb" statement. Following the

```
;Sample Parasol-II Problem
;Bouncing Ball, showing the use of logic statements within a
; differential equation definition.
$dfsb sys
#x = v $$
#v = fs fd + fg - m / $$
;x is position measured from the ground up; v is velocity.
fs =
        ;Note that statements can be continued over as many lines as desired.
        ;This one will be broken up to allow for comments.
        xz x - $dup
        ;this puts two copies of the value of xz-x on the stack. xz is the
        ;distance from that ground that the spring becomes active.
        $iftr ks *      ;If xz-x > 0 this sets fs to k*(xz-x).
        $else $drop 0   ;Otherwise, the spring force is zero (the $drop
                        ;gets rid of the extra (xz-x)).
        $ndif           ;End the if-clause
$$                      ;end of statement defining fs.
.dt = xz 2 * x - $iftr dt1 $else dt2 $ndif $$    ;select the value for the
                                        ;time step on the basis of position
                                        ;relative to the spring. A "buffer"
                                        ;zone is allowed so the contact point
                                        ;is calculated accurately
fd = b v * $chs $$
;Damping force.
endsb
;
;End of simulation block definition.
;Now set values for parameters:
1 &x $ic          ;initial condition for position (v(0) = 0)
1 @m              ;mass of "ball"
0.1 @dt2          ;step size when in free flight
0.01 @dt1         ;step size during "bounce"
200 @ks           ;spring constant
0.05 @b           ;damping coefficient
0.1 @xz           ;active distance of spring
1 @fg             ;gravity force
;Define graphics:
$dffn .hv .t 0 .tmax $$
$dffn .v1 x 0 1 $$
;Set up tabular output to print on a file named bounce.dat
$dffn print .t x v $$
$pfil                   ;redirect output stream
bounce.dat
;
;Run the simulation
$gini axes                      ;initialize graphic device and plot axes
0.1 5 run
$pcls                   ;close the output file
```

Figure ex2.2 Parasol Program Listing

differential equations, there is a section in which parameter values and initial conditions are assigned. Next, the functions that define the graph to be produced are defined. The function .hv, for the horizontal variable, is set so that running time will be the x-axis variable. (The function "time" could also have been used in place of the explicit reference to the predefined variables .t for running time and .tmax for the maximum simulation time.) Only one vertical variable (y-axis) is defined. It is defined with the function .v1 and is the ball's position, x, scaled to a range of 0-1.

The tabular output is defined with the "print" function. The variables .t (running time), x, and v will be tabulated. Since no print interval is specified, the tabular output will be done every time step. The tabular output is directed to the file "bounce.dat" by use of the $pfil command.

Finally, the simulation is actually run. The graphic device is initialized ($gini), axes are drawn, and the simulation is run with a step size of 0.5 and a final time of 5. Note that in this case, the step size will be recomputed in the program, so the specification made in the run command will be ignored. At the completion of the run, the output file is closed ($pcls) so that the operating system will not treat it as a temporary file and erase it when the program is terminated. The graphical output is shown in Figure ex2.3. No graph labels or tick marks are included with the "run" package in order to conserve memory. These are available in the "graph" package and can be loaded with the run package if sufficient memory is available. The tabular output will be used as an example of the graphics operation of Parasol.

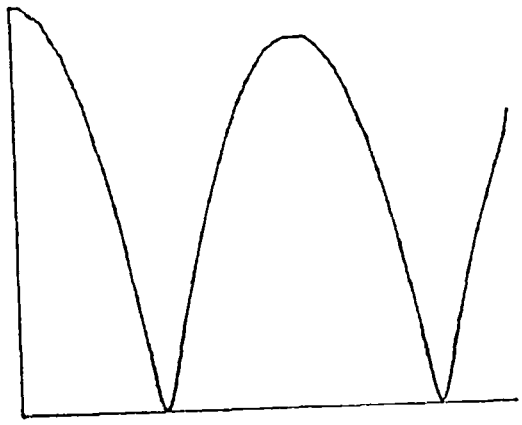


Figure ex2.3 The Ball's Position vs Time

The program as shown was run completely from a file. When Parasol was loaded, assuming that the simulation (run) package was loaded from startp.p, the input stream was immediately directed to this file with a $ifil command. From then on, the operation was completely automatic. To run the problem interactively from the console, the user input would be exactly as shown here. The only difference in appearance would be that Parasol would print prompts to the user when it was expecting input.

Example 3 Velocity Control of a Two-Dimensional Positioning Table

Assume that we have added a microprocessor-based control system to regulate the velocities of the positioning table of example 1. The control object, the table, is simulated by the same differential equation as before, but the controller is a discrete-time device that should be simulated by a difference equation. By making use of the ability to define more than one simulation block, we can simulate both simultaneously.

The "run" package is set up to handle two simulation blocks, the one called "sys" that we have already seen in earlier examples for differential equations, and a second simulation block called "cntrl" for discrete-time equations. In the discrete-time block, the operator "#" is defined as "next-value-of."

The listing for the program is shown in Figure ex3.1. The controllers in the block "cntrl" use the PI algorithm in the "velocity" or difference form. The only other difference from the program of example 1 is that instead of putting in a table of values for the inputs, u1 and u2, they are calculated by the controller and the velocity setpoints, vxset and vyset, are entered with tables. In the section where values are set for constants, the variable ".tsamp" is given a value. This is the sample time for the discrete-time controller and is a reserved variable defined in the "run" package.

The program is run in the same manner as the original problem in example 1. The results are shown in Figure ex3.2.

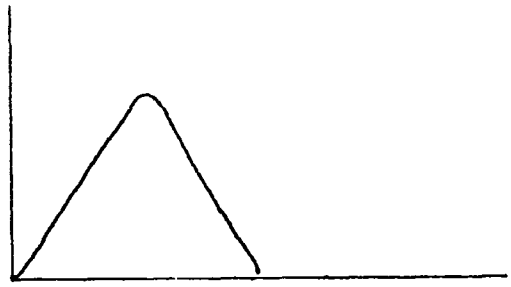


Figure ex3.2 Two-Dimensional Motion with Velocity Control

Example 4. Report Graphics for the Bouncing Ball

In addition to the graph of ball position vs time done at the time of simulation (example 2), a data file containing time, position, and velocity for each computed point was also produced. That file serves as the basis for the report-type graphs that we will do here.

Each line of the file has time, position, and velocity in that order. The variables .xvar and .yvar can be set to indicate which of those values should be plotted on the x-axis and which on the y-axis. In that way, we will be able to produce three graphs, position vs time (a duplicate of the graph done in example 2, but annotated), velocity vs time, and velocity vs position (a phase-plane plot).

Figure ex4.1 shows the listing of the file used to make these graphs. As usual, the same commands could have been given interactively, if desired. In the interactive mode, additional prompts are given

```
;Velocity Control of a Two-Dimensional Positioning Table

;Parasol-II sample problem. This sample illustrates
;the use of multiple simulation blocks. In this case,
;one block, "sys," is a control object (the two-dimensional positioning
;table of the first example) and the other block, named "cntrl"
;is the controller. The function "run" is set up to handle
;two such blocks. The block called "cntrl" is assumed to be
;a discrete-time controller that is active every ".tsamp" time
;units.

$dfsb sys
#x1 = v1 $$              ;define the position state variable for axis #1
#v1 = fm1 b1 v1 * - m1 / $$      ;velocity state equation
u1 = mvx $$              ;input comes from the controller
fm1 = km1 u1 * fm11 $chs fm11 $lim $$
                         ;force applied to table by drive motor --
                         ; u1 is the controlling input. fm11 is the
                         ; limiting value of force that can be applied.
;Same equations for axis #2 --
#x2 = v2 $$              ;define the position state variable for axis #2
#v2 = fm2 b2 v2 * - m2 / $$      ;velocity state equation
u2 = mvy $$
fm2 = km2 u2 * fm21 $chs fm21 $lim $$
                         ;force applied to table by drive motor --
                         ;u1 and u2 will be determined by the velocity
                         ;controller.
endsb            ;end of simulation block

$dfsb cntrl              ;This is the controller block. It is solved
                         ; as a discrete-time difference equation with a
                         ; sample time of .tsmp
ervx = vxset v1 - $$     ;x-axis velocity error
#ervx1 = ervx $$         ;one sample time delay of ervx; ervx(i-1)
ervy = vyset v2 - $$
#ervy1 = ervy $$
#mvx = ervx ervx1 - kpvx * ervx kivx * + mvx + $$
                                    ;PI controller, using
                                    ; velocity algorithm
#mvy = ervy ervy1 - kpvy * ervy kivy * + mvy + $$
endsb            ;End of controller equations

$dffn vxset      ;Table for velocity setpoints
        0 1
        9.5 1    ;Deceleration zone
        10 0
.t 3 $fng $$
$dffn vyset
        0 1
        4.9 1
        5.1 -1
        9.5 -1
        10 0
.t 5 $fng $$
```

```
;Set values for constants (these values can be changed while running the
; problem by typing similar lines with new values).
1 @m1            ;mass of table #1
0.2 @b1          ;damping coefficient -- includes friction and damping
                 ; effects caused by coupling back through the motor.
1 @km1           ;force produced by motor #1
2.5 @m2
0.2 @b2
1.5 @km2
5 @fm11          ;force limit for actuators
5 @fm21
.2 @.tsmp        ;sample interval for controller
4 @kpvx          ;controller gains
0.45 @kivx
4 @kpvy
0.45 @kivy

;Define variables for plotting -- plot x1 vs x2 to get a picture of
;the path. A subsequent run will be used to get the velocity vs time
;curves (or that data could be sent to a file for later plotting).

$dffn .hv x1 0 20 $$     ;horizontal variable and scale
$dffn .v1 x2 0 10 $$     ;vertical variable #1 (the only one in this case)

;Once this file is read in, the user can initialize the graphics device and
; give the run command
```

Figure ex3.1 Parasol program listing

```
;Graphics commmands to draw a notated graph for
; the bouncing ball sample problem.

;These instructions will put three graphs on a single
; sheet -- position vs time, velocity vs time and
; velocity vs position.

8.5 @.hsiz                ;set graph size
6 @.vsiz
.voff 7.5 + @.voff        ;set offset for top graph
sttck                     ;set the x and y axis ticks
0 1 2 3 4 5 $$            ;Time goes from 0 to 5 -- ticks are
                          ; terminated with a $$ since they are
                          ; imbedded in a function
0 .5 1 $$                 ;Position goes from 0 to 1
stlab                     ;setting labels uses the text-in mode
0
1
2
3
4
5

0
0.5
1

                          ;Each label is given on one line. Each group
                          ;of labels is terminated with a null line.
stttl                     ;set the titles
Time

   Position


                          ;Each title is a line, terminated by a null line.
                          ; The extra null line is for the graph title,
                          ; which we aren't using here.
5.06 @.xmax               ;x-axis scale (the largest value in the file for
                          ; time is 5.06)
$gini                     ;Initialize the graphics device
axes tick label title     ;Draw axes, ...
graph                     ;draw the graph .. the next line is the file name
bounce.dat

;Now do graph of velocity vs time
.voff 8 - @.voff          ;move graph down
-1.5 @.ymin               ;set y-axis scales
1.5 @.ymax
sttck
0 1 2 3 4 5 $$
-1.5 -1 -.5 0 .5 1 1.5 $$
stlab
0
1
2
```

```
3
4
5

-1.5
-1
-.5
0
.5
1
1.5

stttl
Time

   Velocity


3 @.yvar                    ;set the y-axis variable to #3 in the file
axes tick label title graph
bounce.dat

;Now do the phase-plane plot, position on the x-axis, velocity on the y-axis
.voff 8 - @.voff            ;move graph down
1 @.xmax                    ;position scale
2 @.xvar                    ;x-axis variable is now #2
sttck                       ;set ticks
0 .5 1 $$
-1.5 -1 -.5 0 .5 1 1.5 $$
stlab
0
0.5
1

-1.5
-1
-.5
0
.5
1
1.5

stttl
Position

   Velocity


axes tick label title graph
bounce.dat
```

Figure ex4.1 Graphics Commands to Draw Bouncing Ball Position, Velocity and Phase-Plane Graphs (continued from previous page)

to indicate the formats for tick and label inputs. The beginning part of the file sets the graph size (which, in this case, is different from the standard) and moves the graph up to the top of the page so that we can fit three graphs on a single sheet (the graph is being made with a Houston DMP-4 pen plotter). The first graph uses the default values of 1 and 2 for .xvar and .yvar to get the position vs time graph. Ticks, labels, and titles are set with the commands sttck, stlab, and stttl, the x-axis scale is set to correspond to the maximum value in the file (5.06), the graphics device is initialized ($gini), the axes, etc., are drawn, and, finally, the graph itself is drawn with the "graph" command. The line following the "graph" command specifies the data file.

The procedure is repeated for the remaining two graphs. To run this program, Parasol is loaded with the "graph" package. Any initializing that has to be done for the specific graphic device is then done. This file is then invoked with a call to $ifil and the graphs are drawn without any further operator intervention. The resulting set of three graphs is shown in Figure ex4.2.

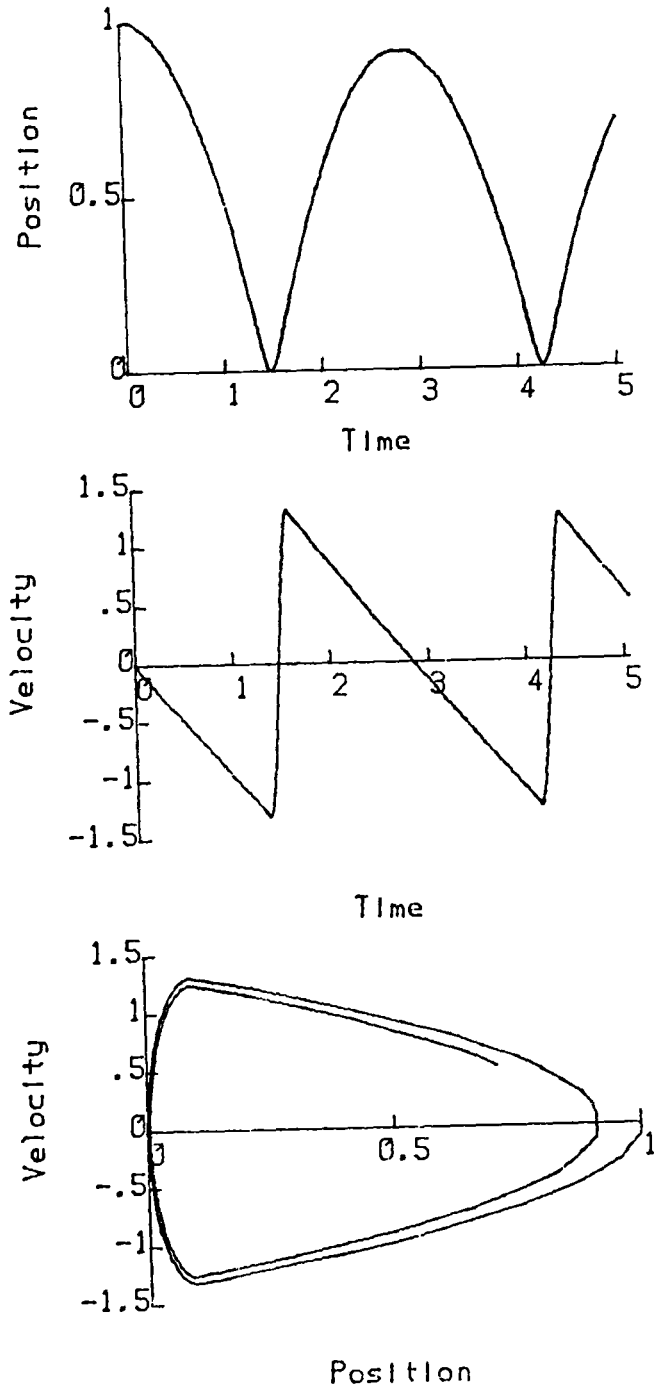


Figure ex4.2 Notated Graph for Bouncing Ball Simulation

Controlled Ecological Life Support Systems (CELSS):
A Bibliography of CELSS Documents Published as NASA Reports

1. Johnson, Emmett J.: Genetic Engineering Possibilities for CELSS: A Bibliography and Summary of Techniques. (NASA Purchase Order No. A73308B.) NASA CR-166306, March 1982.

2. Hornberger, G.M.; and Rastetter, E.B.: Sensitivity Analysis as an Aid in Modelling and Control of (Poorly-Defined) Ecological Systems. (NASA Purchase Order No. A77474.) NASA CR-166308, March 1982.

3. Tibbitts, T.W.; and Alford, D.K.: Controlled Ecological Life Support Systems: Use of Higher Plants. NASA CP-2231, 1982.

4. Mason, R.M.; and Carden, J.L.: Controlled Ecological Life Support Systems: Research and Development Guidelines. NASA CP-2232, 1982.

5. Moore, B.; and R.D. MacElroy: Controlled Ecological Life Support Systems: Biological Problems. NASA CP-2233, 1982.

6. Aroeste, H.: Application of Guided Inquiry System Technique (GIST) to Controlled Ecological Life Support Systems (CELSS). (NASA Purchase Order Nos. A82705B and A89697B.) NASA CR-166312, January 1982.

7. Mason, R.M.: CELSS Scenario Analysis: Breakeven Calculation. (NASA Purchase Order No. A70035B.) NASA CR-166319, April 1980.

8. Hoff, J.E.; Howe, J.M.; and Mitchell, C.A.: Nutritional and Cultural Aspects of Plant Species Selection for a Controlled Ecological Life Support System. (NASA Grant Nos. NSG-2401 and 2404.) NASA CR-166324, March 1982.

9. Averner, M.: An Approach to the Mathematical Modelling of a Controlled Ecological Life Support System. (NASA Contract No. NAS2-10133.) NASA CR-166331, August 1981.

10. Maguire, B.: Bibliography of Human Carried Microbes' Interaction with Plants. (NASA Purchase Order No. A77042.) NASA CR-16630, August 1980.

11. Howe, J.M.; and Hoff, J.E.: Plant Diversity to Support Humans in a CELSS Ground-Based Demonstrator. (NASA Grant No. NSG-2401.) NASA CR-166357, June 1982.

12. Young, G.: A Design Methodology for Nonlinear Systems Containing Parameter Uncertainty: Application to Nonlinear Controller Design. (NASA Cooperative Agreement No. NCC 2-67) NASA CR-166358, May 1982.

13. Karel, M.: Evaluation of Engineering Foods for Controlled Ecological Life Support Systems (CELSS). (NASA Contract No. NAS 9-16008.) NASA CR-166359, June 1982.

14. Stahr, J.D.; Auslander, D.M.; Spear, R.C.; and Young, G.E.: An Approach to the Preliminary Evaluation of Closed-Ecological Life Support System (CELSS) Scenarios and Control Strategies. (NASA Cooperative Agreement No. NCC 2-67) NASA CR-166368, July 1982.

15. Radmer, R.; Ollinger, O.; Venables, A.; Fernandez, E.: Algal Culture Studies Related to a Closed Ecological Life Support System (CELSS). (NASA Contract No. NAS 2-10969) NASA CR-166375, July 1982.

16. Auslander, D.M.; Spear, R.C.; and Young, G.E.: Application of Control Theory to Dynamic Systems Simulation. (NASA Cooperative Agreement No. NCC 2-67) NASA CR- 166383 , August 1982.

1 Report No	2 Government Accession No	3 Recipient's Catalog No
NASA CR-166383		

4 Title and Subtitle	5 Report Date
Application of Control Theory to Dynamic Systems Simulation	August 1982
	6 Performing Organization Code

7 Author(s)	8 Performing Organization Report No
Auslander, D.M.,* R.C. Spear, and G.E. Young	

9 Performing Organization Name and Address	10 Work Unit No
Departments of Mechanical Engineering and *Biomedical and Environmental Health Sciences University of California, Berkeley, CA 94720	T5992
	11 Contract or Grant No NCC 2-67

12 Sponsoring Agency Name and Address	13 Type of Report and Period Covered
National Aeronautics and Space Administration Washington, D.C. 24056	Contractors Report
	14 Sponsoring Agency Code 199-60-62

15 Supplementary Notes
Robert D. MacElroy, Technical Monitor, Mail Stop 239-10, Ames Research Center, Moffett Field, CA 94035 (415) 965-5573 FTS 448-5573. The 16th in a series of CELSS reports.

16 Abstract

This report contains 3 papers which consider the application of control theory to dynamic systems simulation. These articles contain theory and methodology applicable to controlled ecological life support systems (CELSS). Discussed are spatial effects on system stability, design of control systems with uncertain parameters, and an interactive computing language (PARASOL-II) designed for dynamic system simulation, report-quality graphics, data acquisition, and simple real-time control.

17 Key Words (Suggested by Author(s))	18 Distribution Statement
CELSS, Life Support Systems Control Theory Dynamic Systems System Stability Simulation Language (PARASOL-II)	Unclassified - Unlimited STAR Category 54

19 Security Classif (of this report)	20 Security Classif (of this page)	21 No of Pages	22 Price*
Unclassified	Unclassified	63	

*For sale by the National Technical Information Service Springfield, Virginia 22161

End of Document